


Unidade VI:

Ordenação Interna - Heapsort



PUC Minas

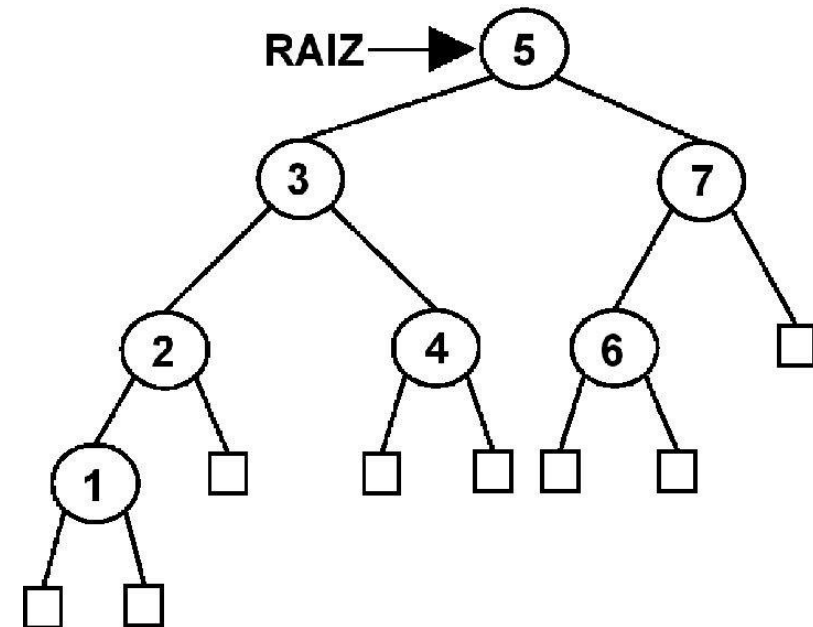
Adaptação dos slides elaborados pelo Instituto de Ciências Exatas e
Informática - Departamento de Ciência da Computação

- **Definição de Heap** 
- Funcionamento básico
- Algoritmo
- Análise do número de comparações e movimentações

Introdução

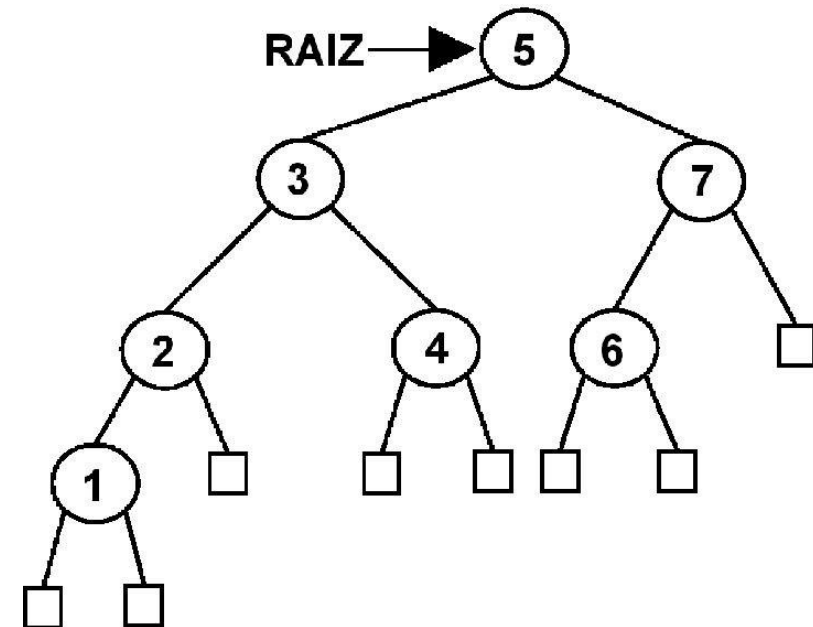
- O Heapsort é **um algoritmo de seleção** que encontra o maior elemento em uma lista, troca-o com o último e repete o processo
- Sua diferença em relação ao Algoritmo de Seleção é que o Heapsort utiliza um Heap Invertido para selecionar o maior elemento de forma eficiente
- Neste momento, precisamos conhecer os conceitos de árvore e heap

- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices



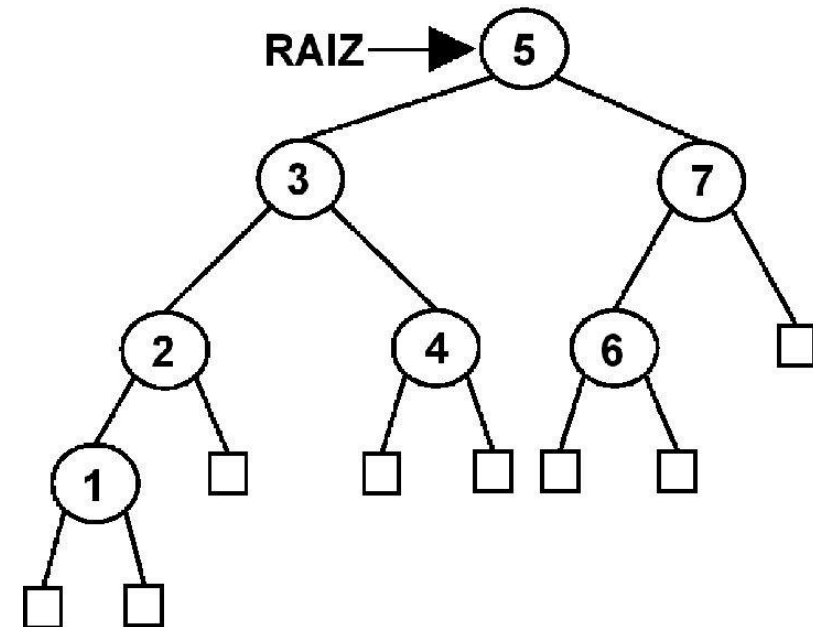
- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices

O nó 5 é denominado nó raiz e ele está no nível 0



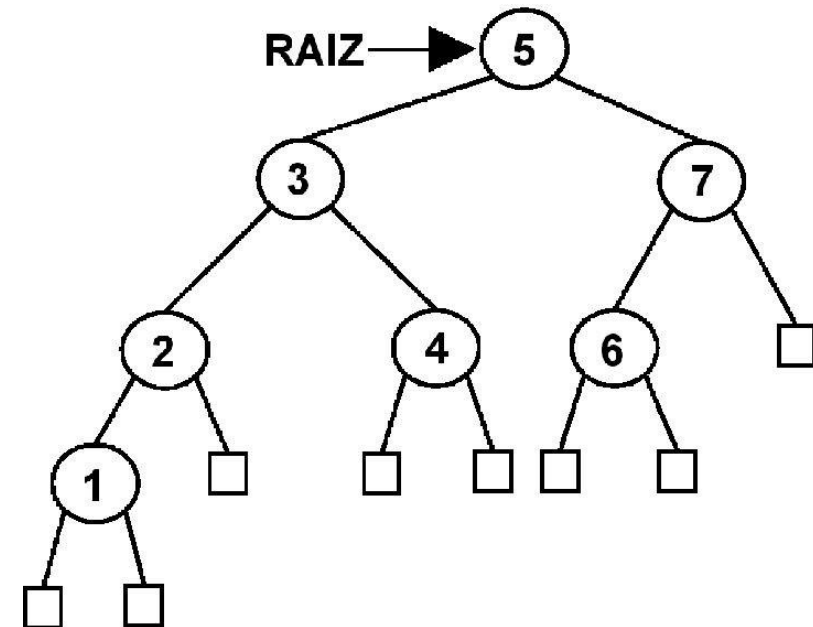
- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices

Os nós 3 e 7 são filhos do 5 e esse é pai dos dois primeiros



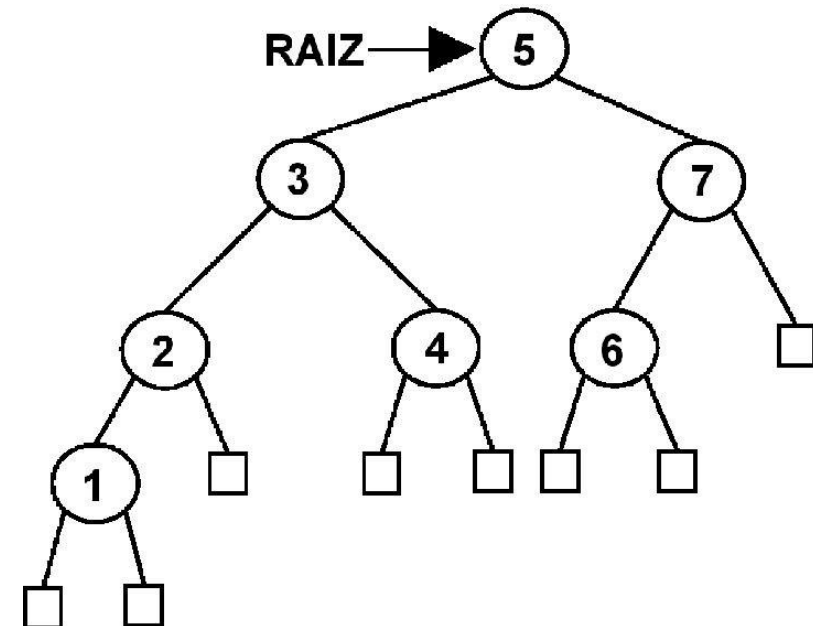
- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices

Um nó com filho(s) é chamado de nó interno e outro sem, de folha

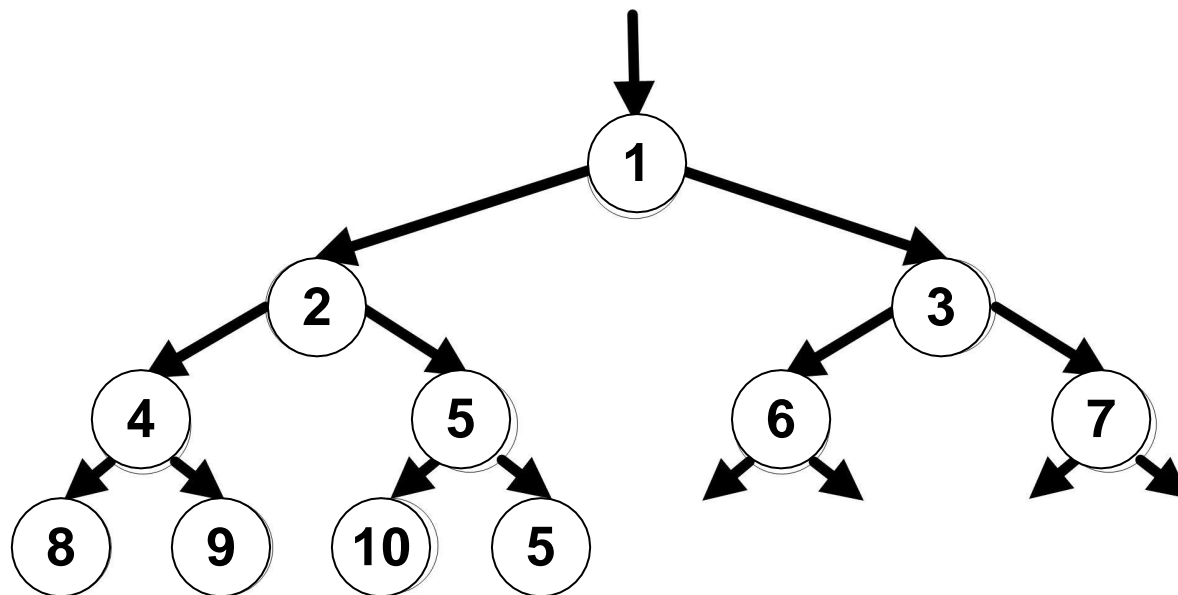


- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outro de arcos (arestas) que conectam os vértices

Nosso exemplo é uma árvore binária, pois cada nó tem no máximo dois filhos



- Árvore binária em que cada nó é menor ou igual que seus filhos, fazendo com que a raiz tenha o menor valor
- Suas folhas ocupam um ou dois níveis sendo que o penúltimo é completo e as folhas do último nível se agrupam o mais à esquerda possível

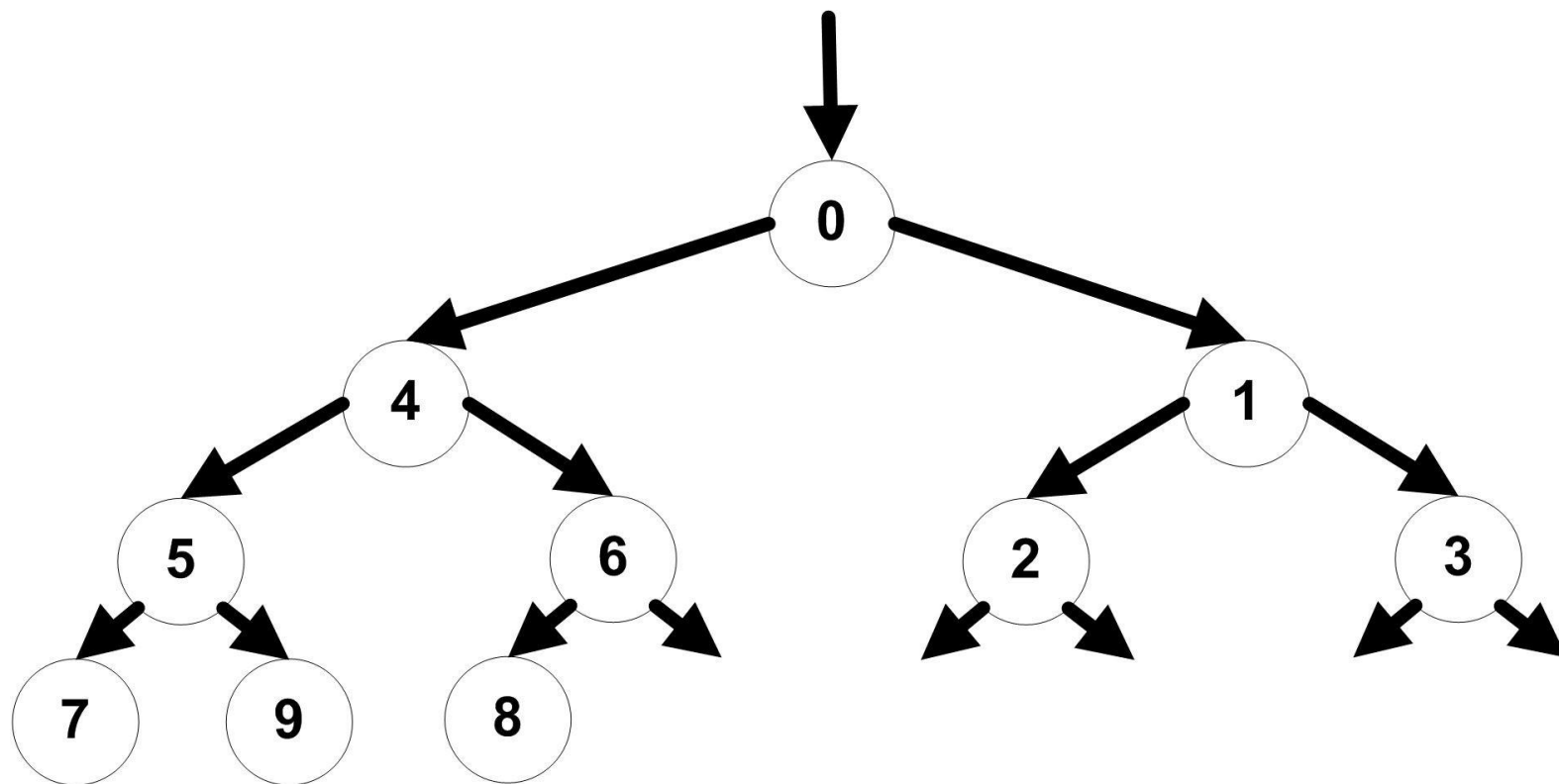


Exercício Resolvido

- Mostre um heap com os elementos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9

Exercício Resolvido

- Mostre um heap com os elementos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9



Heap Invertido

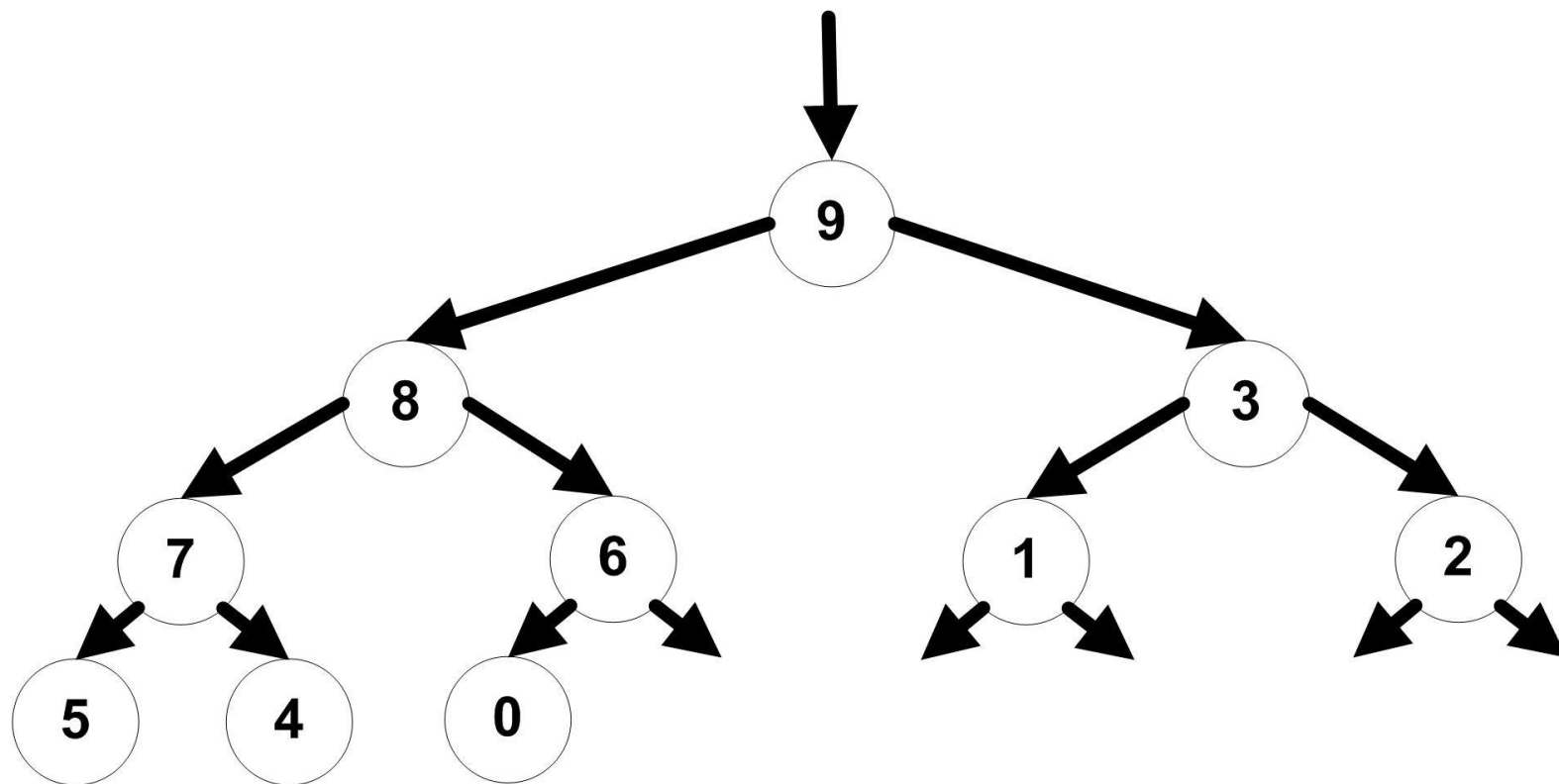
- Árvore binária em que cada nó é **maior** ou igual que seus filhos, fazendo com que a raiz tenha o **maior** valor
- Suas folhas ocupam um ou dois níveis sendo que o penúltimo é completo e as folhas do último nível se agrupam o mais à esquerda possível

Exercício Resolvido

- Mostre um heap invertido com os elementos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9

Exercício Resolvido

- Mostre um heap invertido com os elementos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9



Consideração

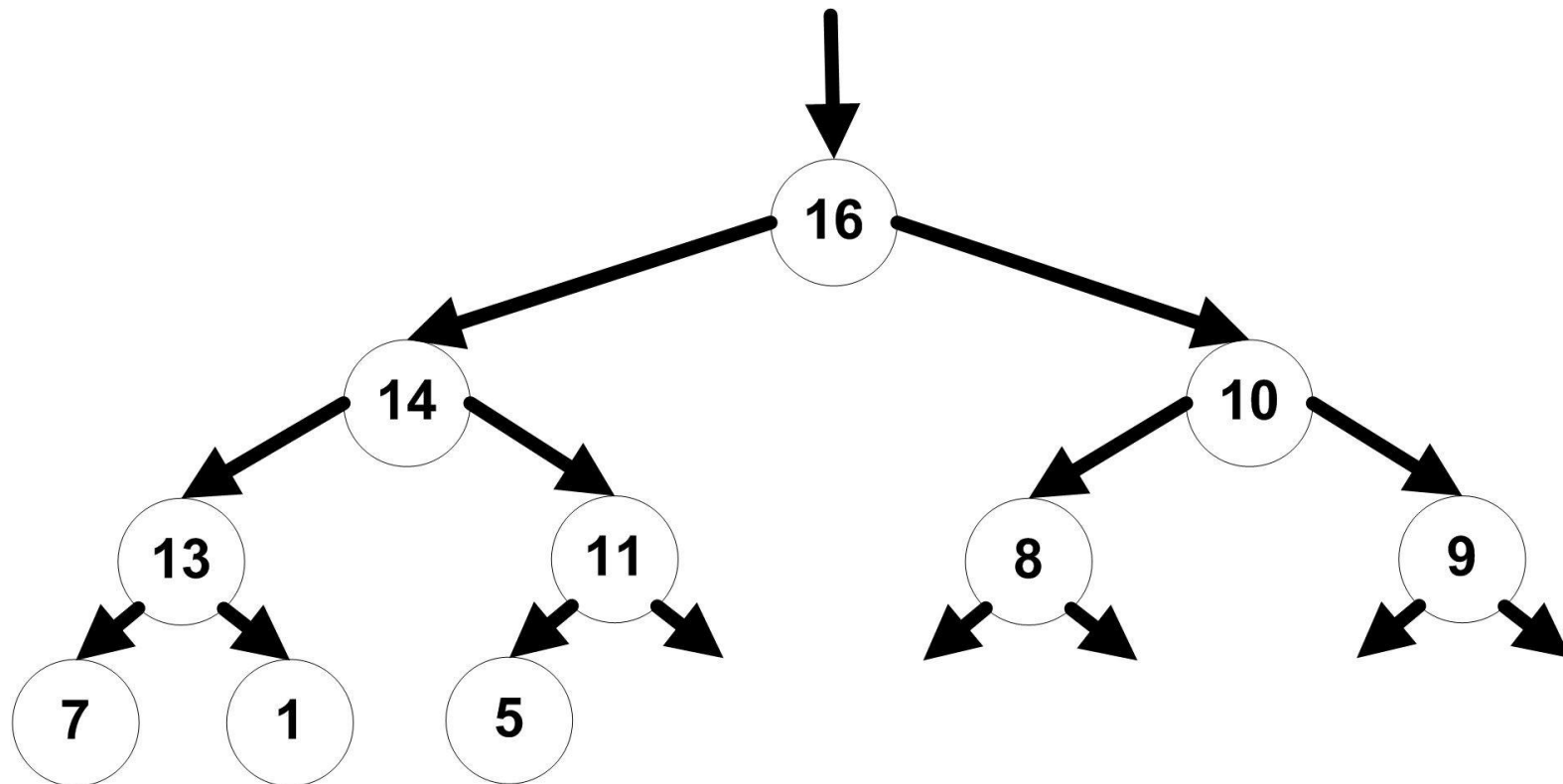
- A partir deste ponto, neste material, a palavra heap será usada para designar o **heap invertido**

Representação do Heap em um *Array*

- Como representar um heap usando um *array*?
- Afinal, estamos apresentando um algoritmo para ordenar *arrays*...

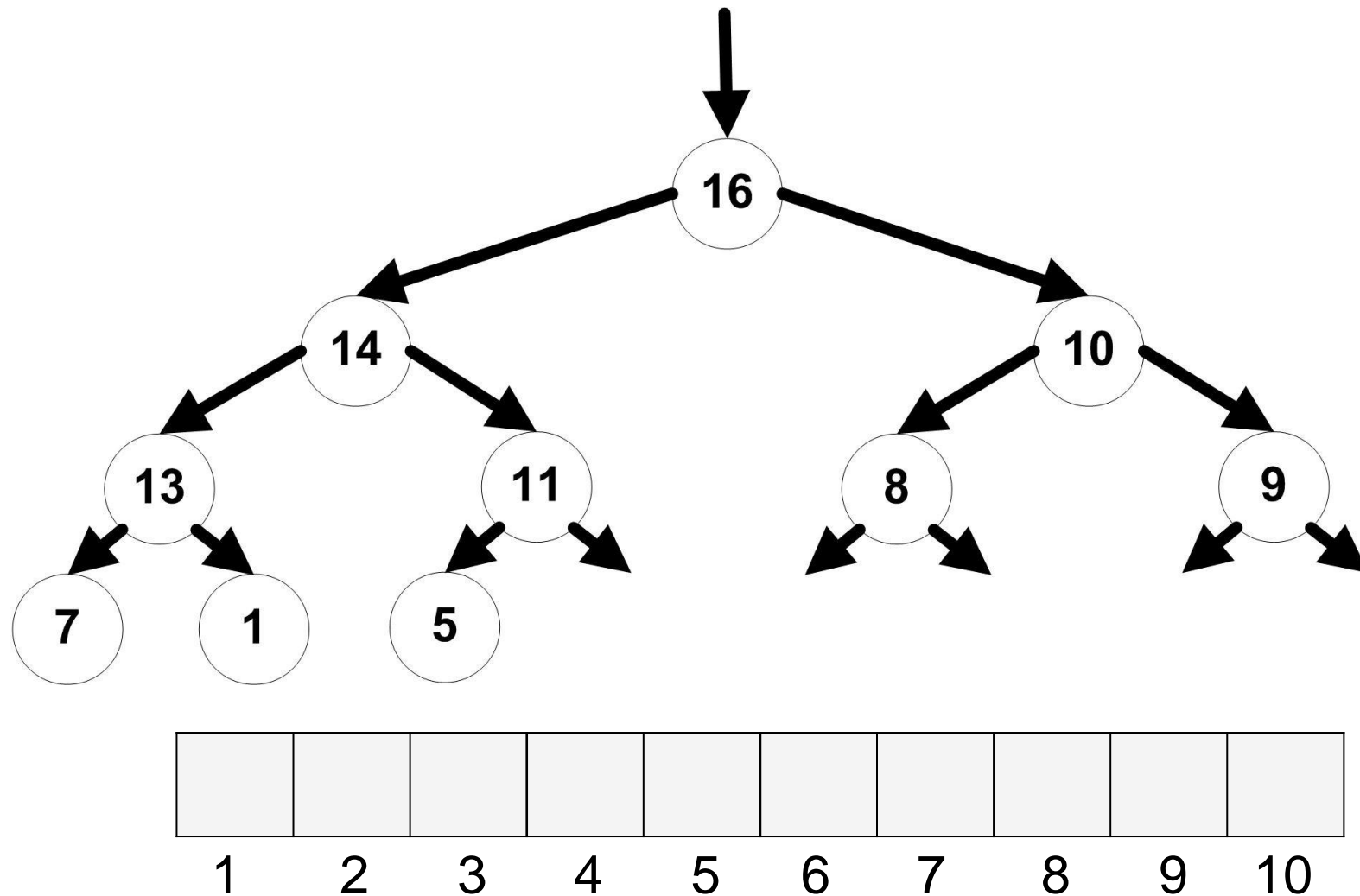
Exercício Resolvido

- Represente o heap abaixo em um *array*



Exercício Resolvido

- Represente o heap abaixo em um *array*

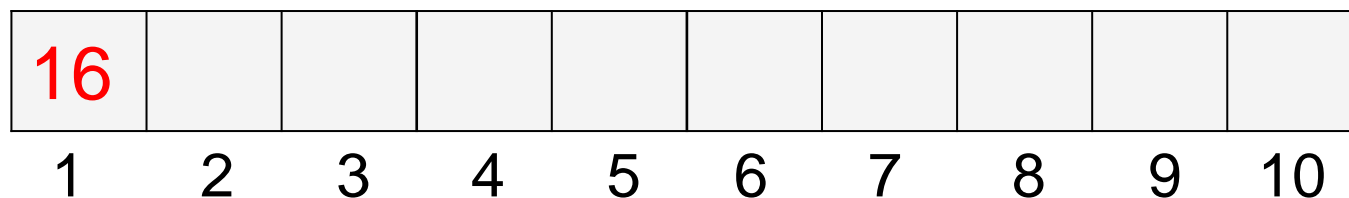
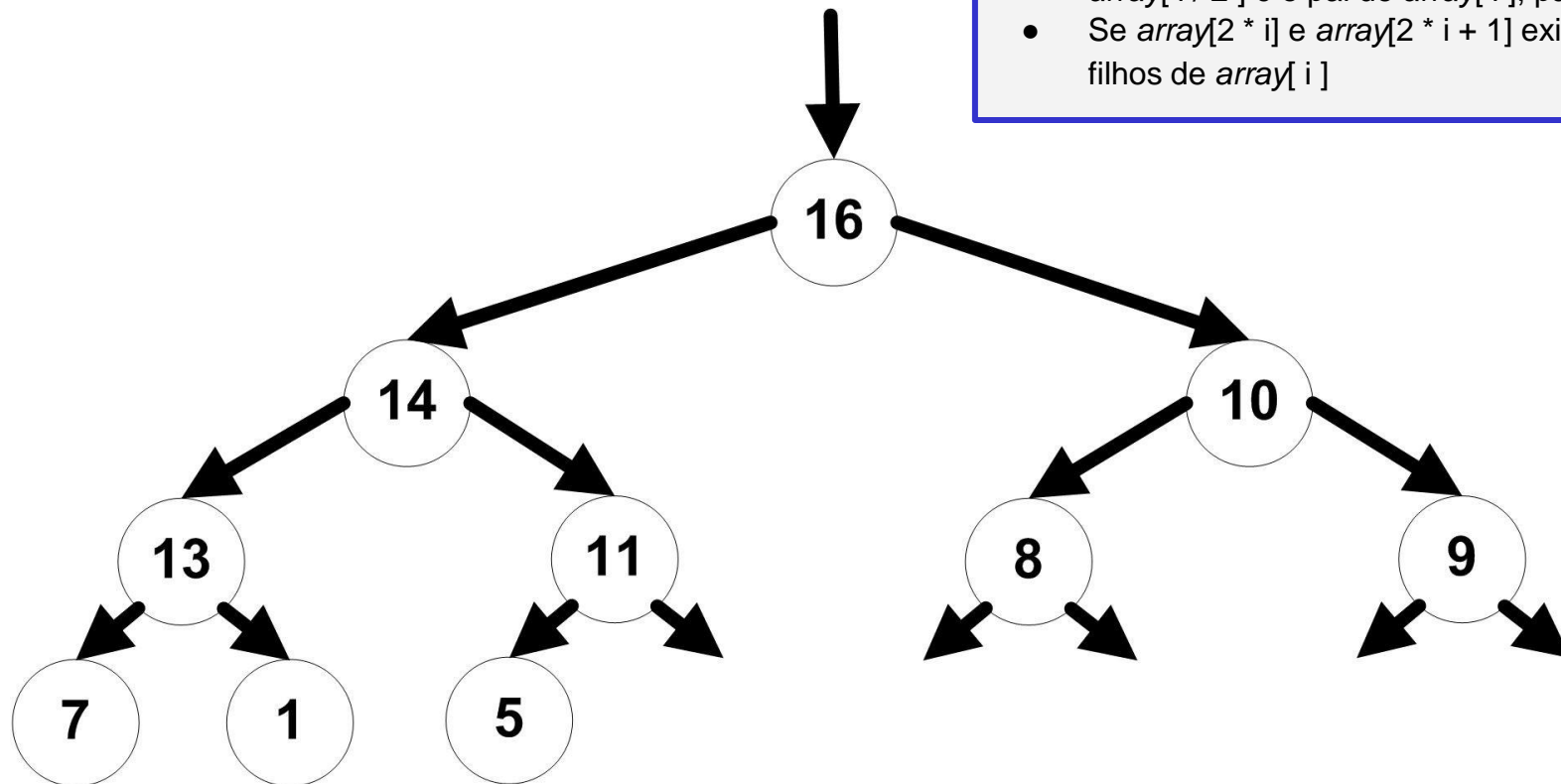


Exercício Resolvido

- Represente o heap abaixo em um *array*

COLA

- $array[1]$ é a raiz
- $array[i/2]$ é o pai de $array[i]$, para $i > 1$
- Se $array[2*i]$ e $array[2*i+1]$ existem, eles são filhos de $array[i]$

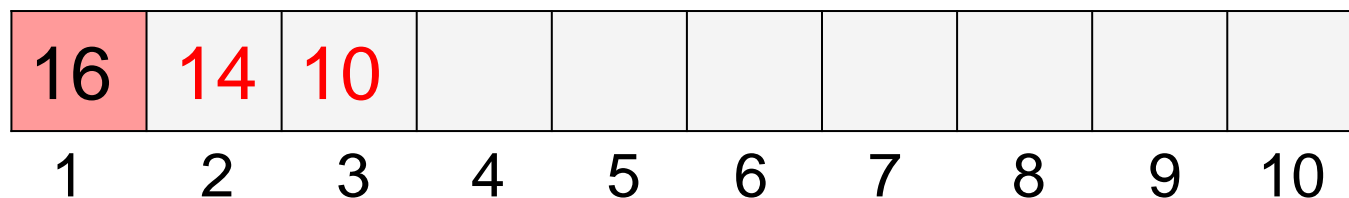
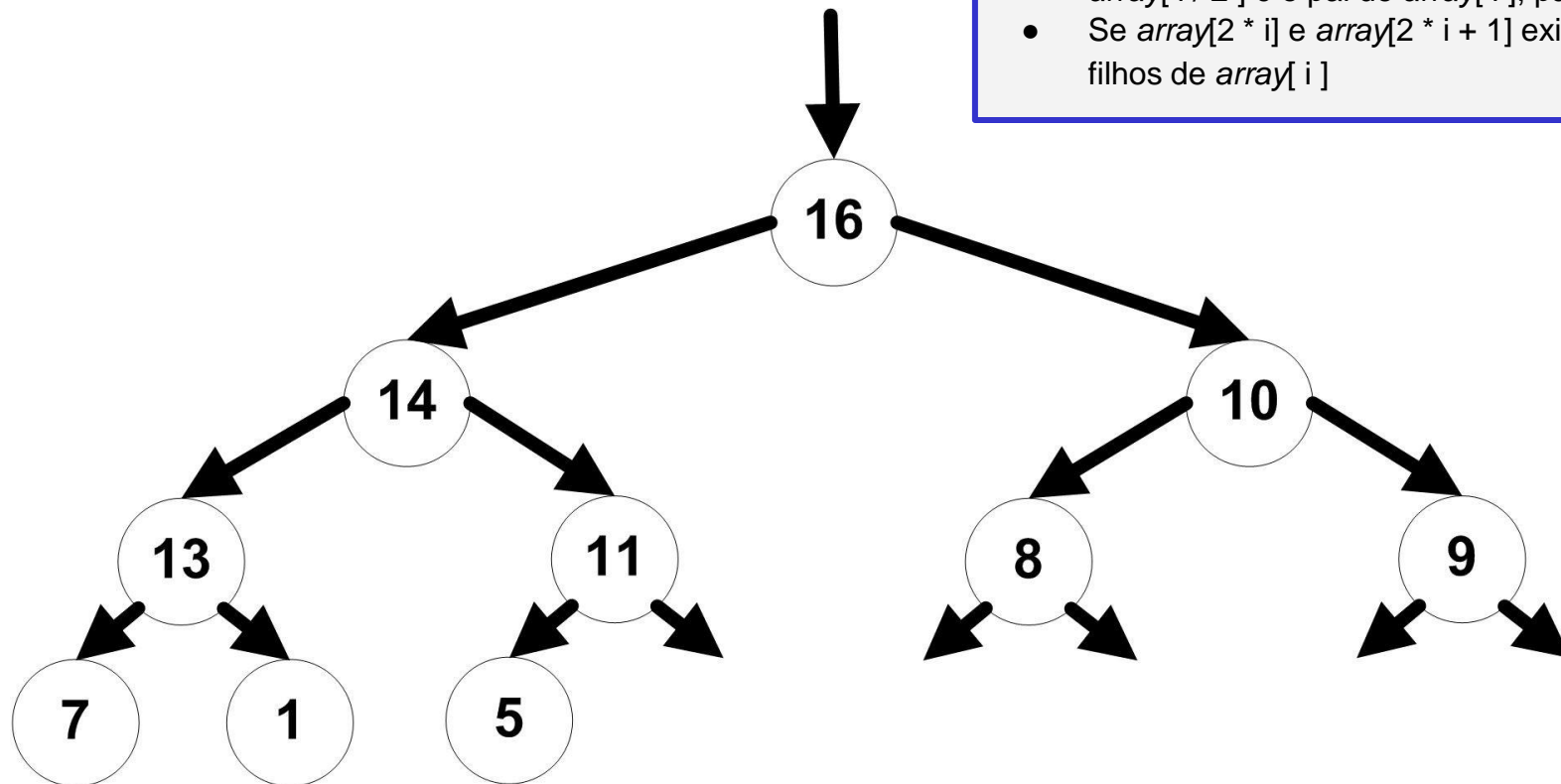


Exercício Resolvido

- Represente o heap abaixo em um *array*

COLA

- $array[1]$ é a raiz
- $array[i/2]$ é o pai de $array[i]$, para $i > 1$
- Se $array[2*i]$ e $array[2*i+1]$ existem, eles são filhos de $array[i]$

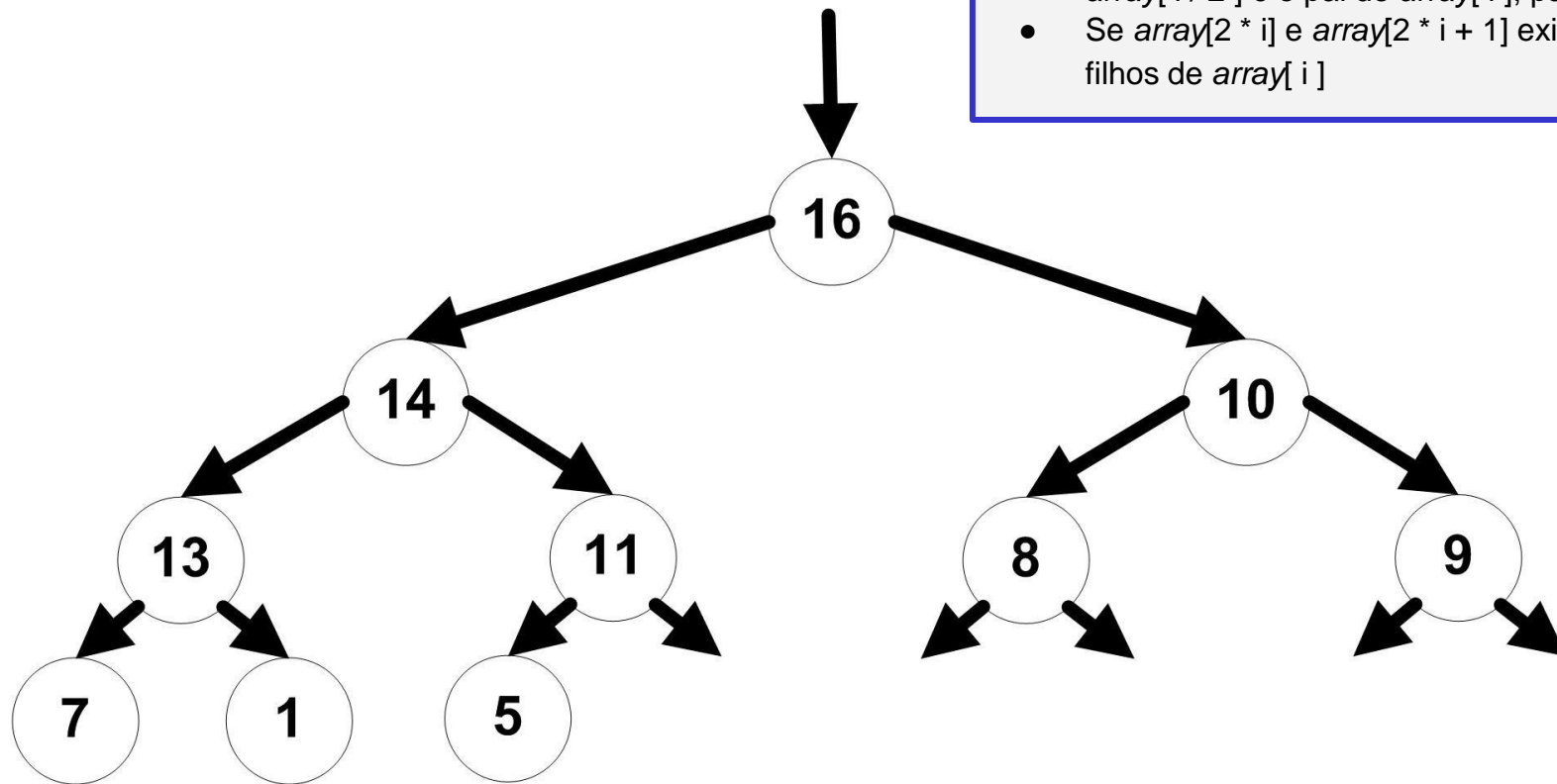


Exercício Resolvido

- Represente o heap abaixo em um *array*

COLA

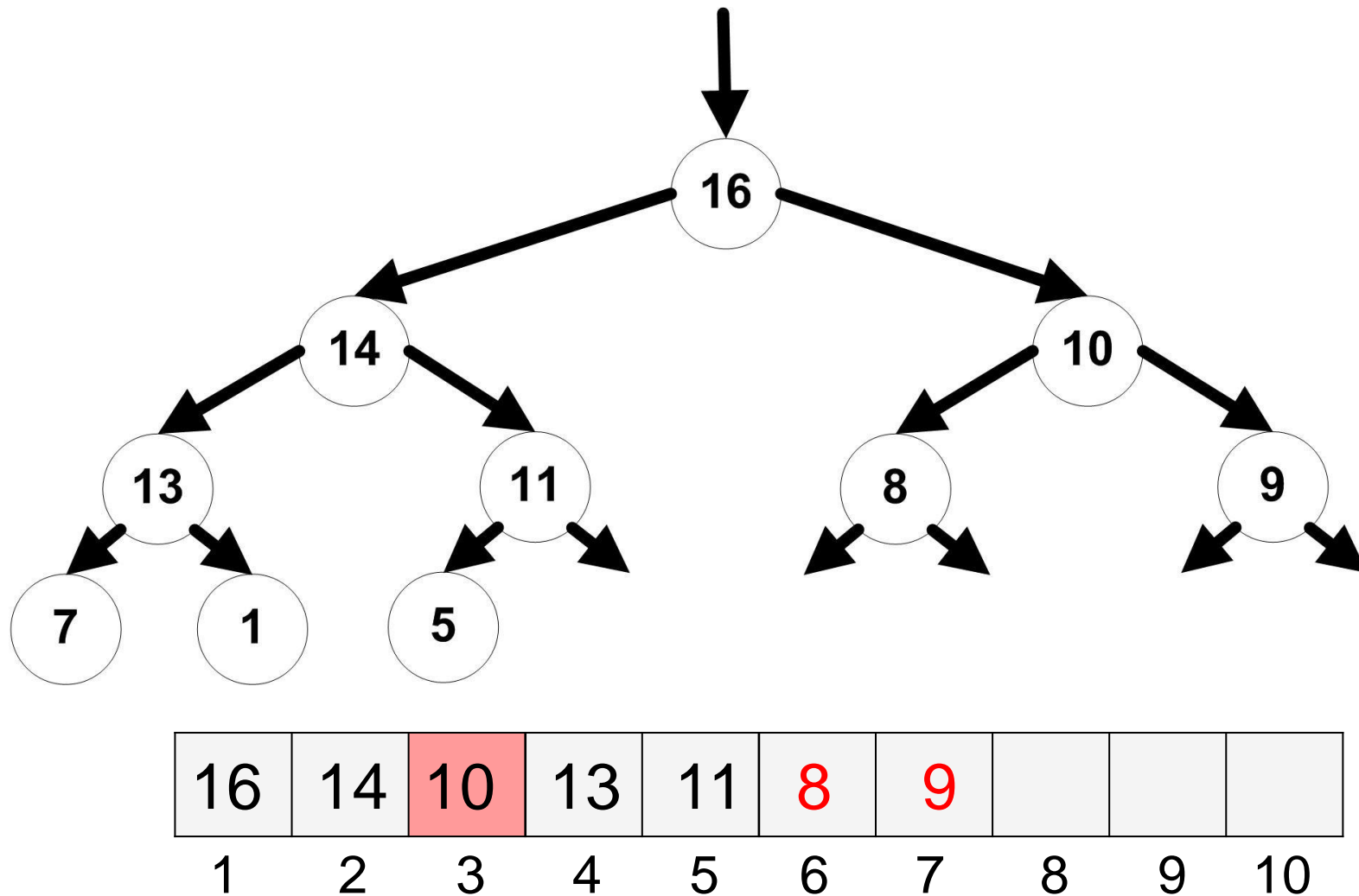
- $array[1]$ é a raiz
- $array[i/2]$ é o pai de $array[i]$, para $i > 1$
- Se $array[2*i]$ e $array[2*i+1]$ existem, eles são filhos de $array[i]$



16	14	10	13	11					
1	2	3	4	5	6	7	8	9	10

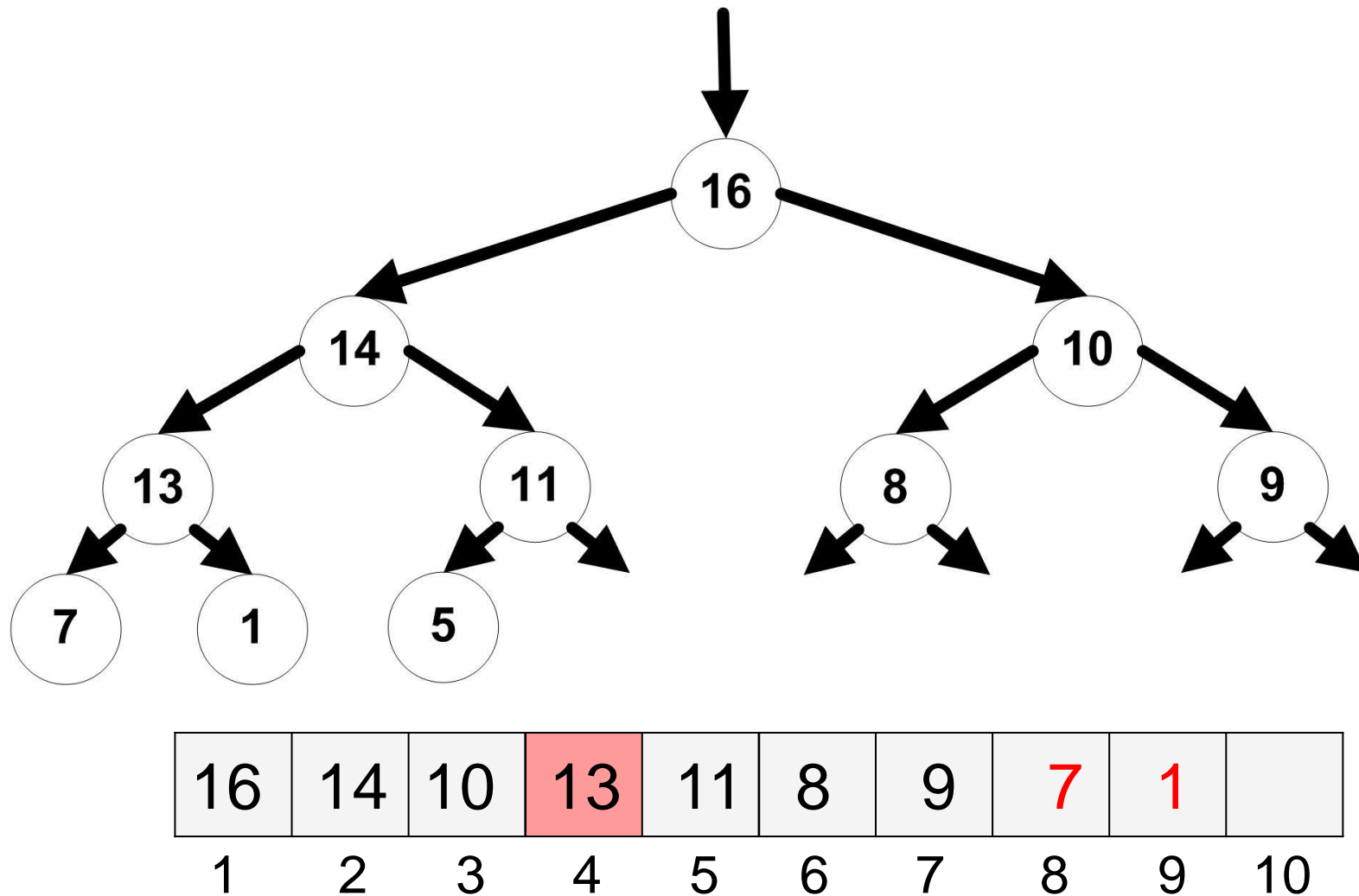
Exercício Resolvido

- Represente o heap abaixo em um *array*



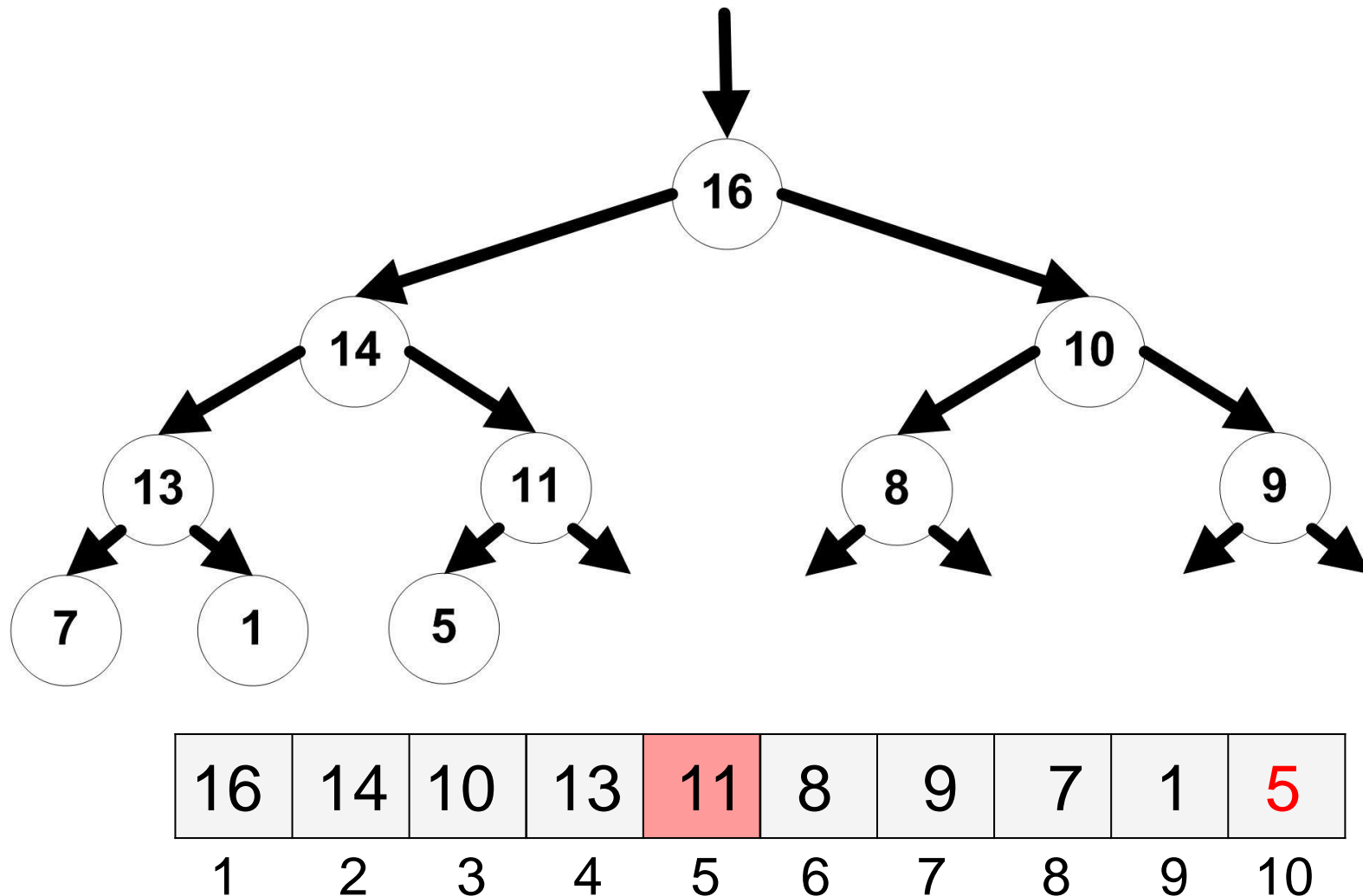
Exercício Resolvido

- Represente o heap abaixo em um *array*



Exercício Resolvido

- Represente o heap abaixo em um *array*



Representação do Heap em um *Array*

• Podemos representar qualquer árvore binária em um *array* fazendo:

- $array[1]$ é a raiz
- $array[i / 2]$ é o pai de $array[i]$, para $i > 1$
- Se $array[2 * i]$ e $array[2 * i + 1]$ existem, eles são filhos de $array[i]$
- Se i é maior que $(n / 2)$, $array[i]$ é uma folha

Representação do Heap em um *Array*

• Podemos representar qualquer árvore binária em um *array* (*0-based index*) fazendo:

- $array[0]$ é a raiz
- $array[(i-1)/2]$ é o pai de $array[i]$, para $i > 0$
- Se $array[2*i + 1]$ e $array[2*i + 2]$ existem, eles são filhos de $array[i]$
- Se i é maior ou igual que $(n/2)$, $array[i]$ é uma folha

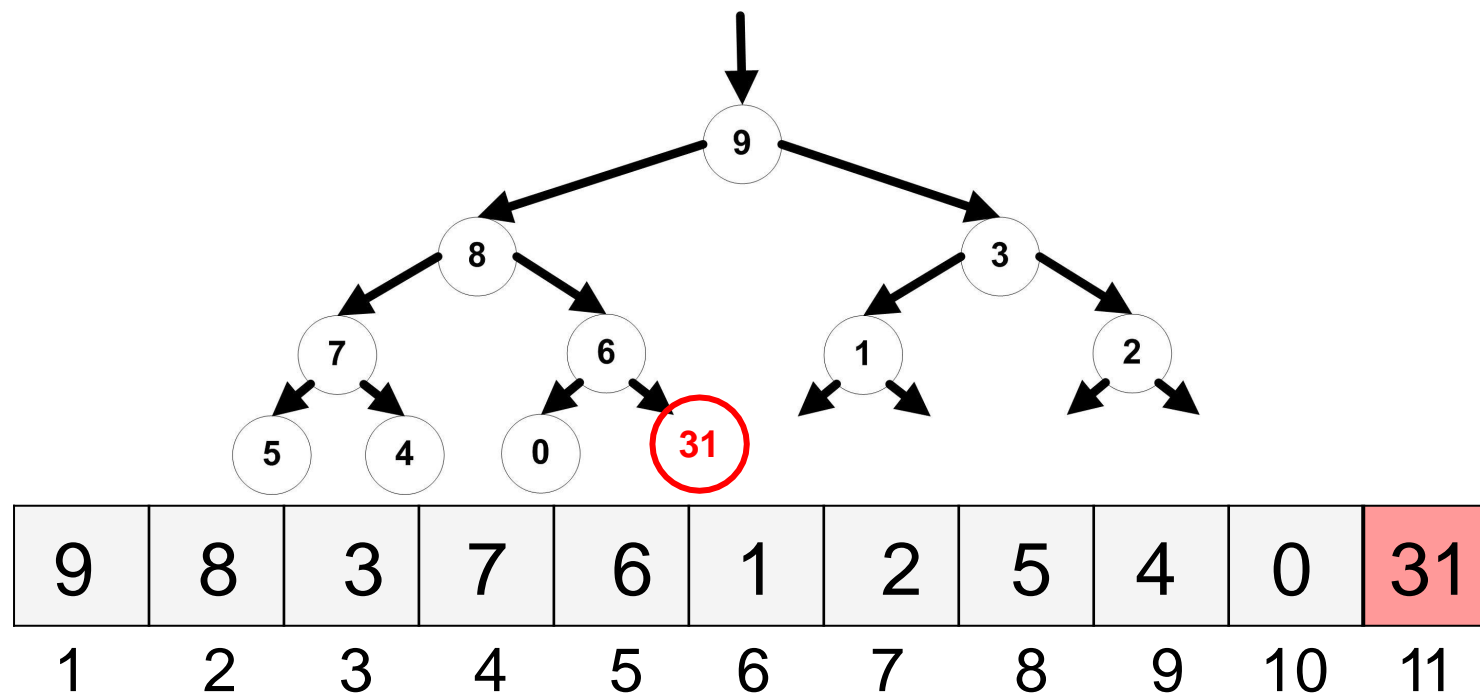
Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

9	8	3	7	6	1	2	5	4	0	31
1	2	3	4	5	6	7	8	9	10	11

Exercício Resolvido

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31




Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

9	8	3	7	6	1	2	5	4	0	31
1	2	3	4	5	6	7	8	9	10	11



Exercício Resolvido

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

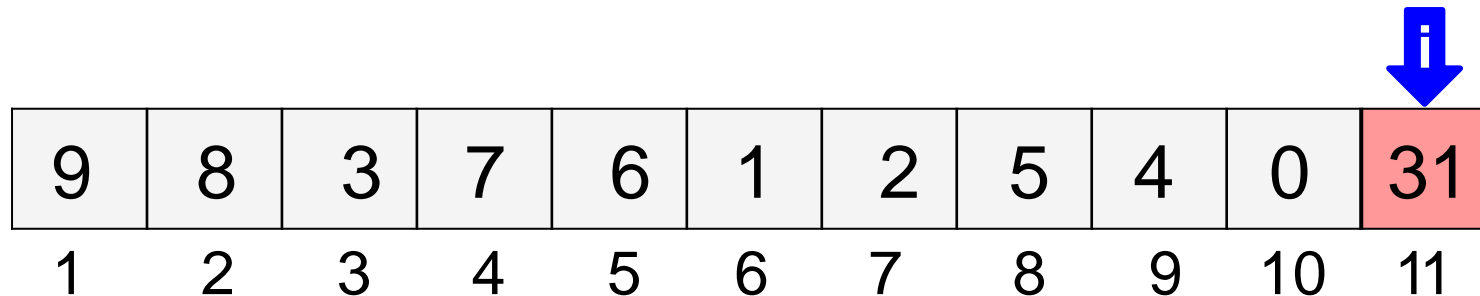
```
void Swap(int[] array, int i, int j)  
{  
    int temp = array[i];  
    array[i] = array[j];  
    array[j] = temp;  
}
```

9	8	3	7	6	1	2	5	4	0	31
1	2	3	4	5	6	7	8	9	10	11

Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```




Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o (k+1)-ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

true: $11 > 1 \ \&\& \text{array}[11] > \text{array}[5]$

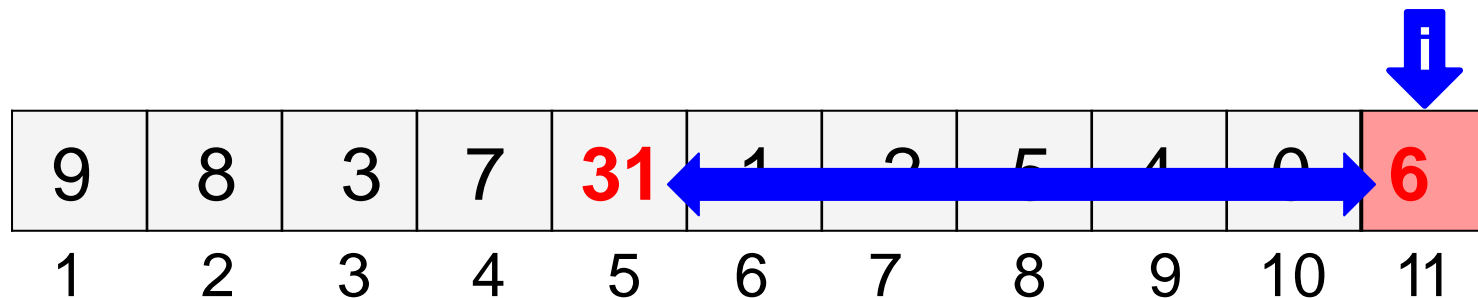
9	8	3	7	6	1	2	5	4	0	31
1	2	3	4	5	6	7	8	9	10	11



Exercício Resolvido

- Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

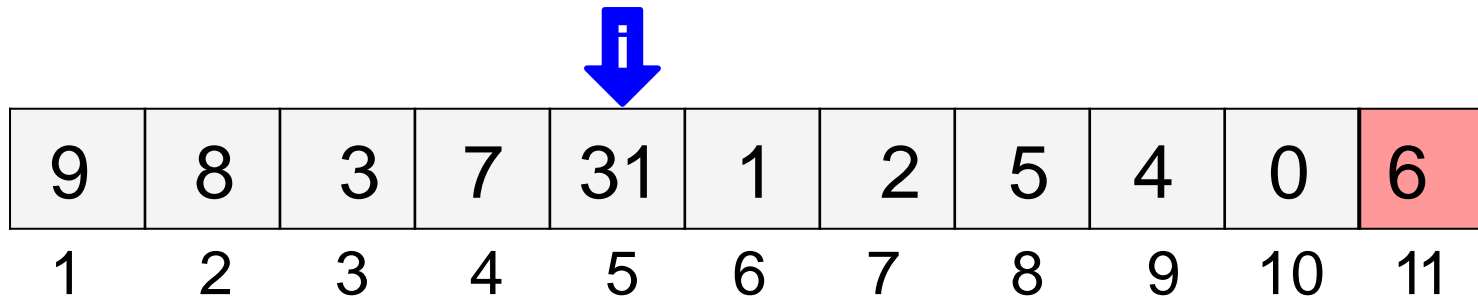
```
void construir(int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

true: $5 > 1 \ \&\& \text{array}[5] > \text{array}[2]$

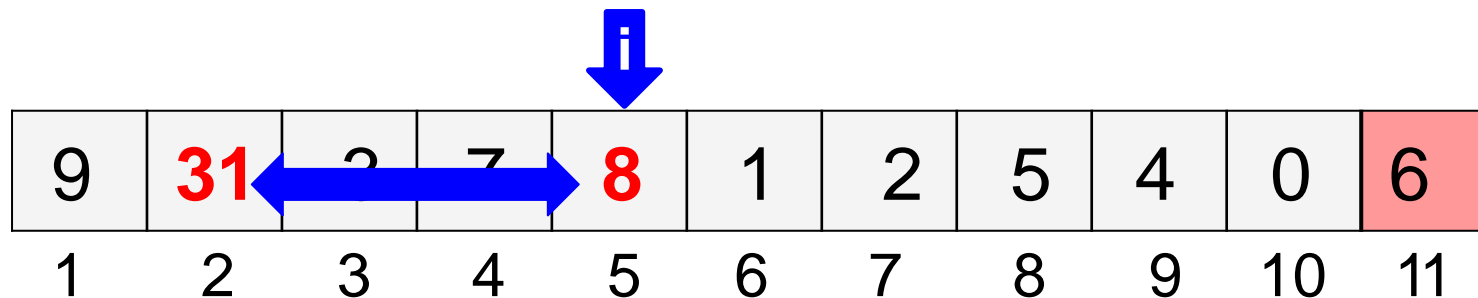


9	8	3	7	31	1	2	5	4	0	6
1	2	3	4	5	6	7	8	9	10	11

Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

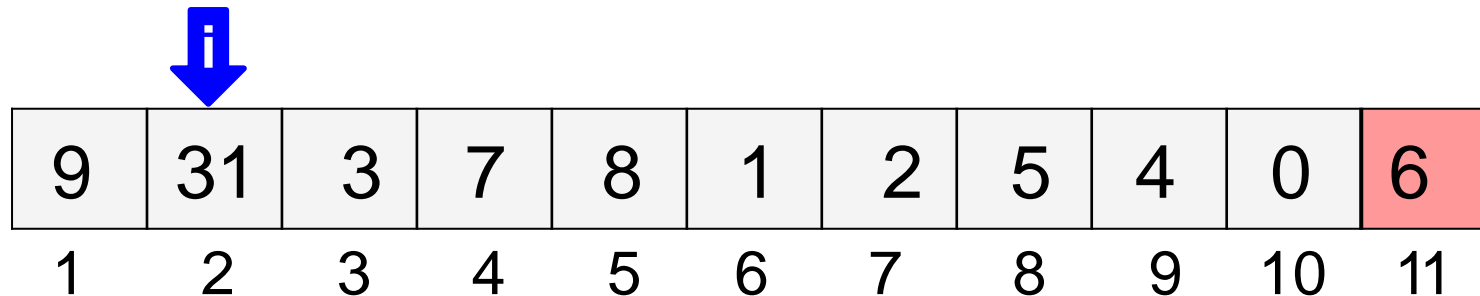
```
void Construir(int[] array, int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

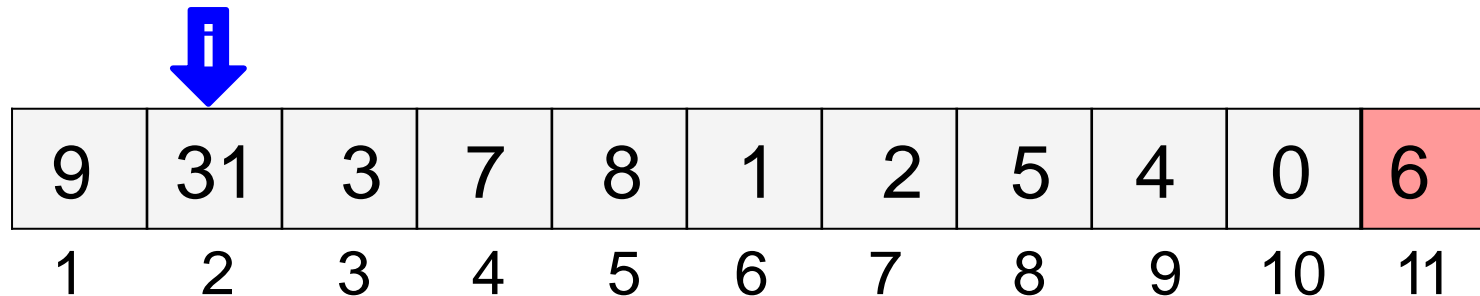


Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

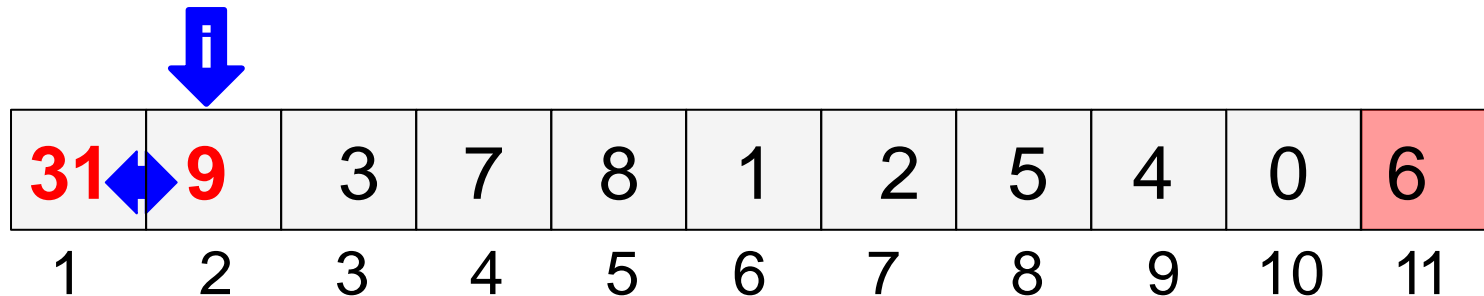
true: $2 > 1 \ \&\& \text{array}[2] > \text{array}[1]$



Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

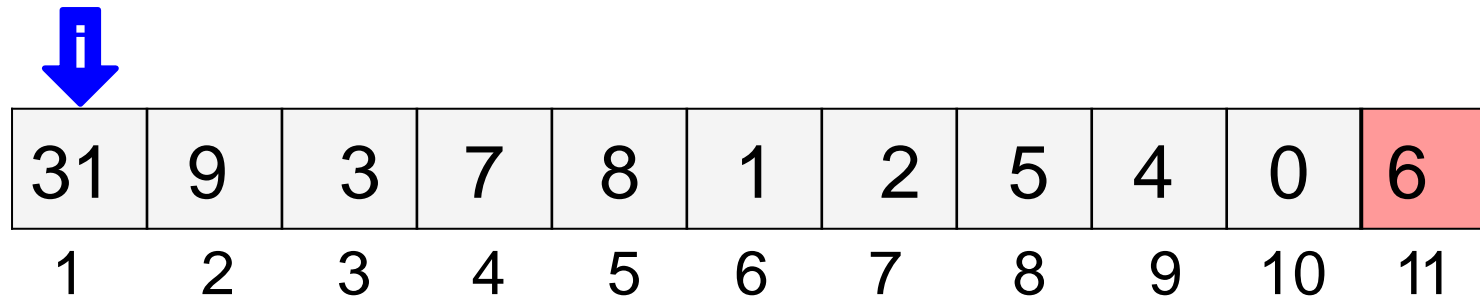
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

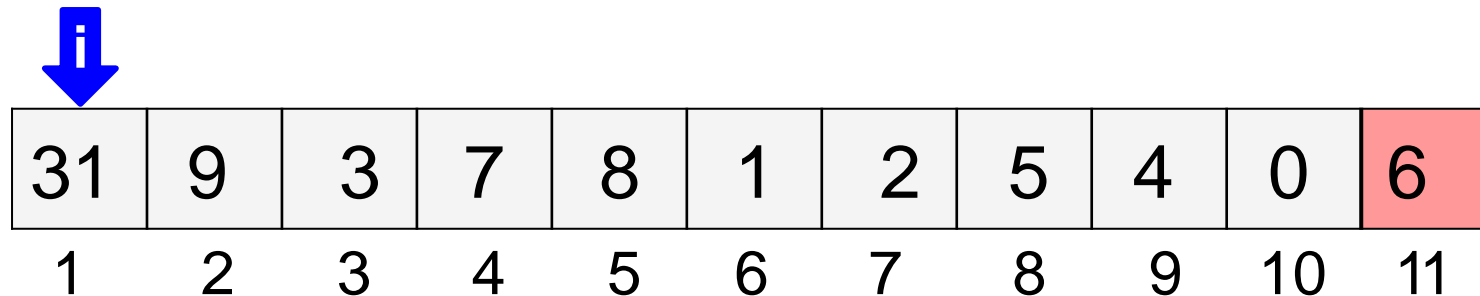


Exercício Resolvido

•Sabendo que os k primeiros elementos de um *array* estão organizados no formato de heap, faça um método que insira o $(k+1)$ -ésimo elemento do *array* no heap. Por exemplo, no *array* abaixo, temos que k é igual a 10 e o elemento a ser inserido no heap é o 31


```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

false: $1 > 1 \ \&\& \dots$



Operações Possíveis com um Heap

- Construção
- Inserção de um novo elemento
- Remoção do elemento com a maior chave
- Remoção de um elemento qualquer
- ...

- Definição de Heap
- **Funcionamento básico** 
- Algoritmo
- Análise do número de comparações e movimentações

Funcionamento Básico

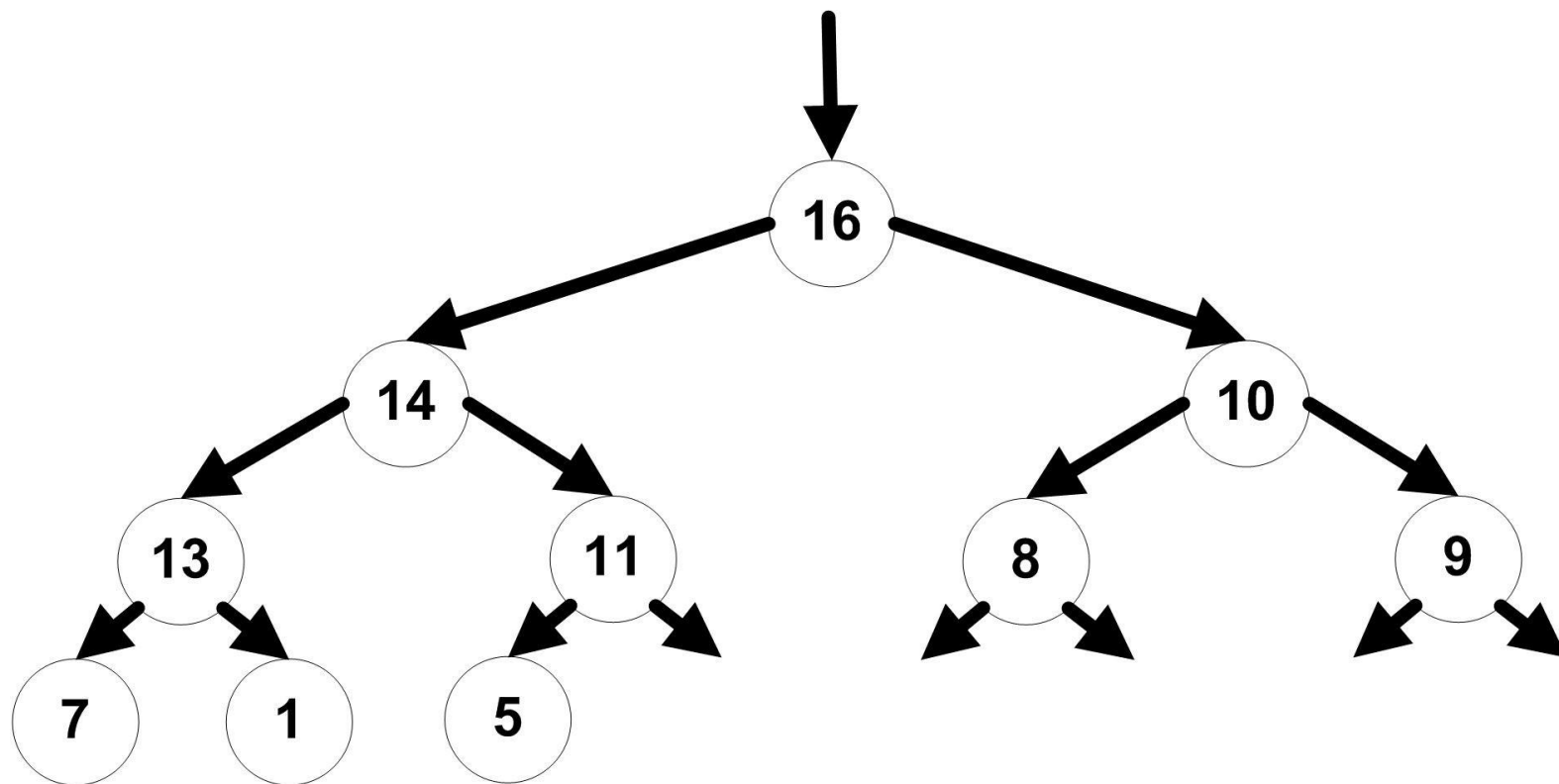
- Construa o heap inserindo sistematicamente cada um dos elementos do *array*
- Remova sistematicamente cada elemento do heap, reconstrua o heap e insira o elemento removido na posição do *array* imediatamente seguinte ao tamanho corrente do heap (isso será feito, trocando o elemento da raiz/primeira posição com o da última posição do heap)
- A seguir, veja os princípios de inserção e remoção no heap

Princípio de Inserção

- Crie uma nova folha (contendo o novo elemento) no último nível do heap.
Se esse estiver completo, recomece um novo nível
- Se o novo elemento for maior que seu pai, troque-os e realize o mesmo processo para o pai, avô, bisavô e, assim, sucessivamente, até que todos os pais sejam maiores que seus filhos

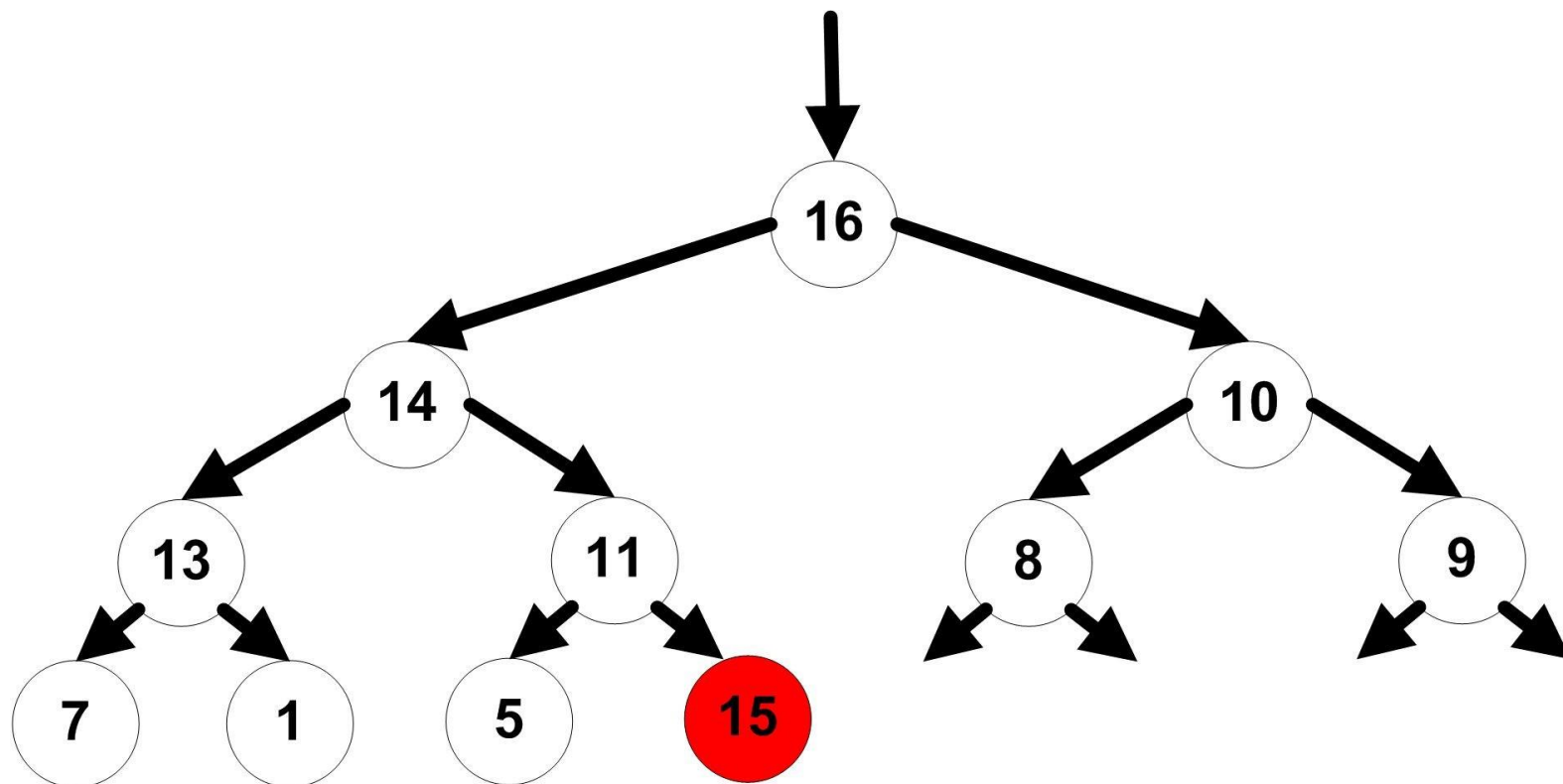
Exercício Resolvido

- Inserir o 15 no heap abaixo



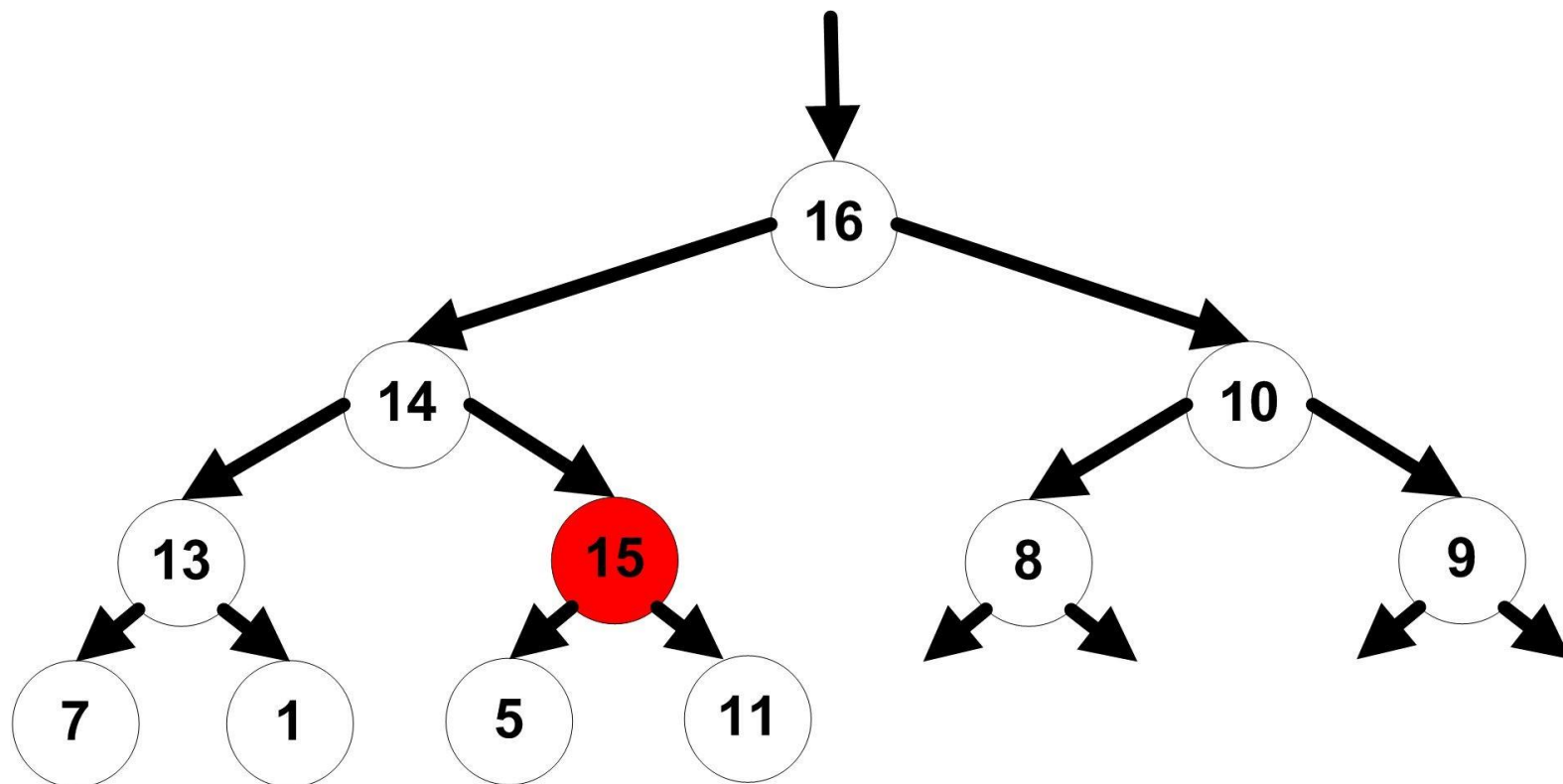
Exercício Resolvido

- Inserir o 15 no heap abaixo



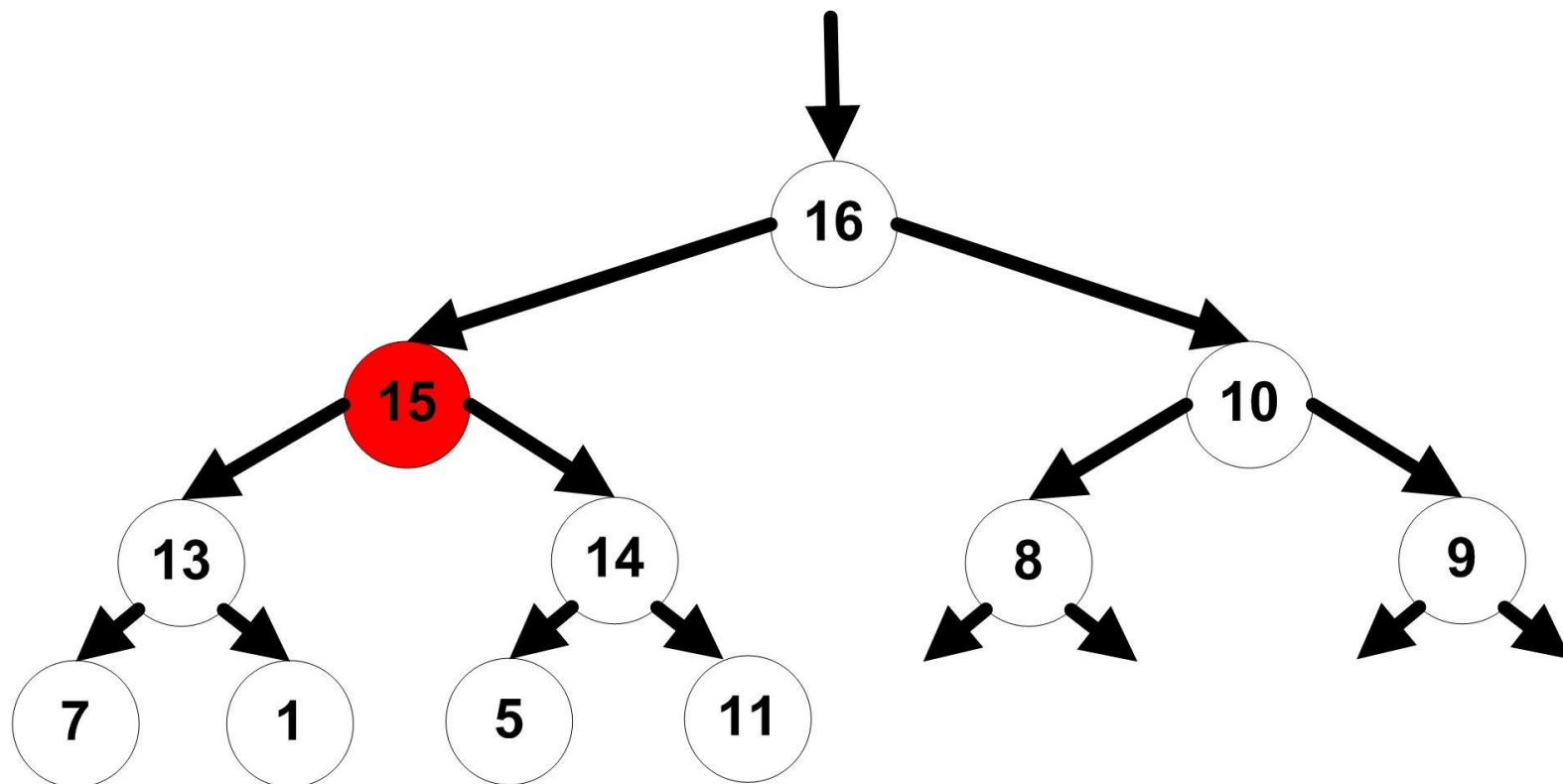
Exercício Resolvido

- Inserir o 15 no heap abaixo



Exercício Resolvido

- Inserir o 15 no heap abaixo

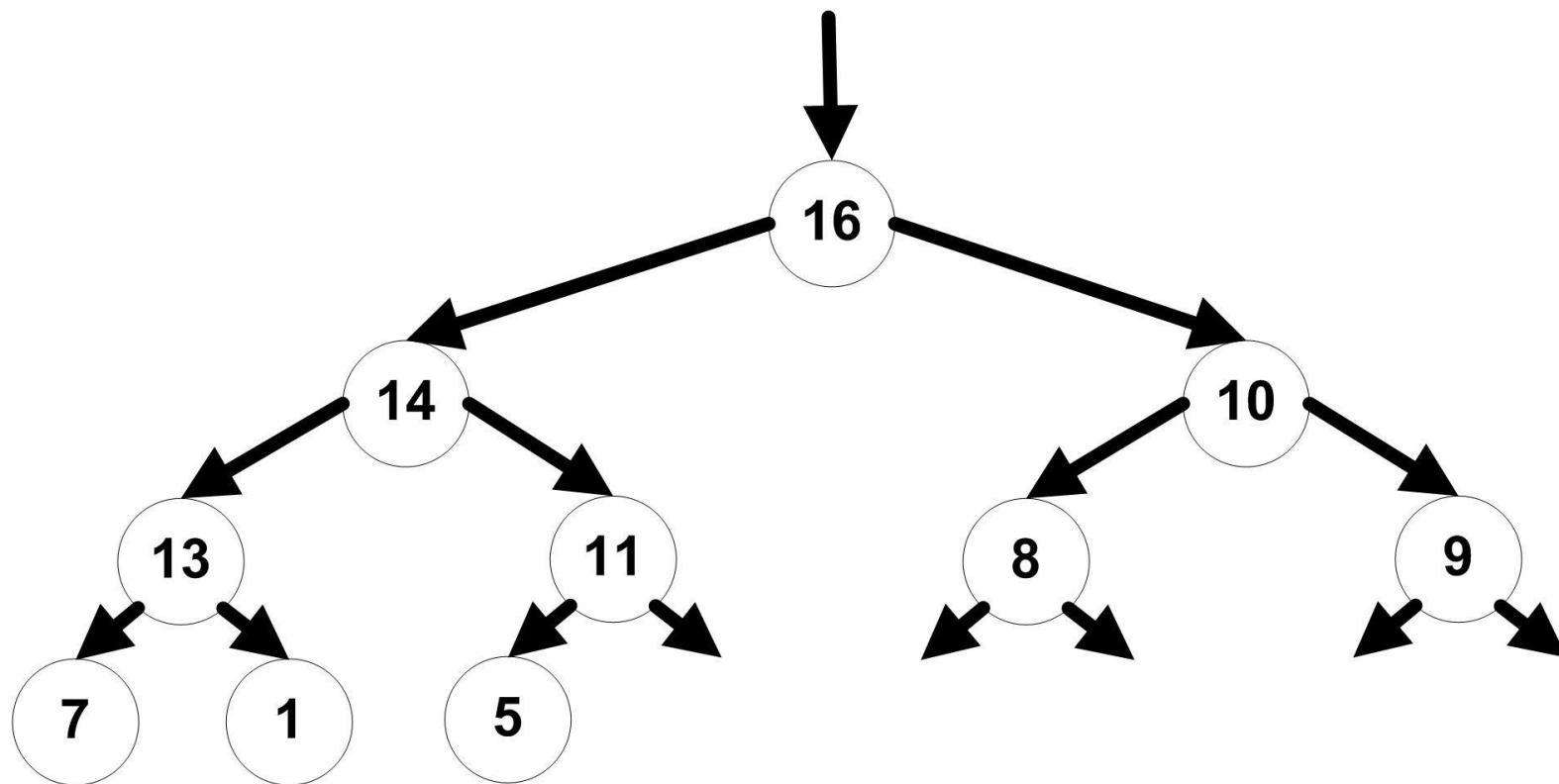


Princípio de Remoção

- Armazene o elemento da raiz em uma variável temporária
- Substitua o elemento da raiz pelo da última folha do último nível
- Remova a última folha do último nível
- Troque o elemento da raiz com o de seu maior filho
- Repita o passo anterior para o filho com elemento trocado até que todos os pais sejam maiores que seus filhos

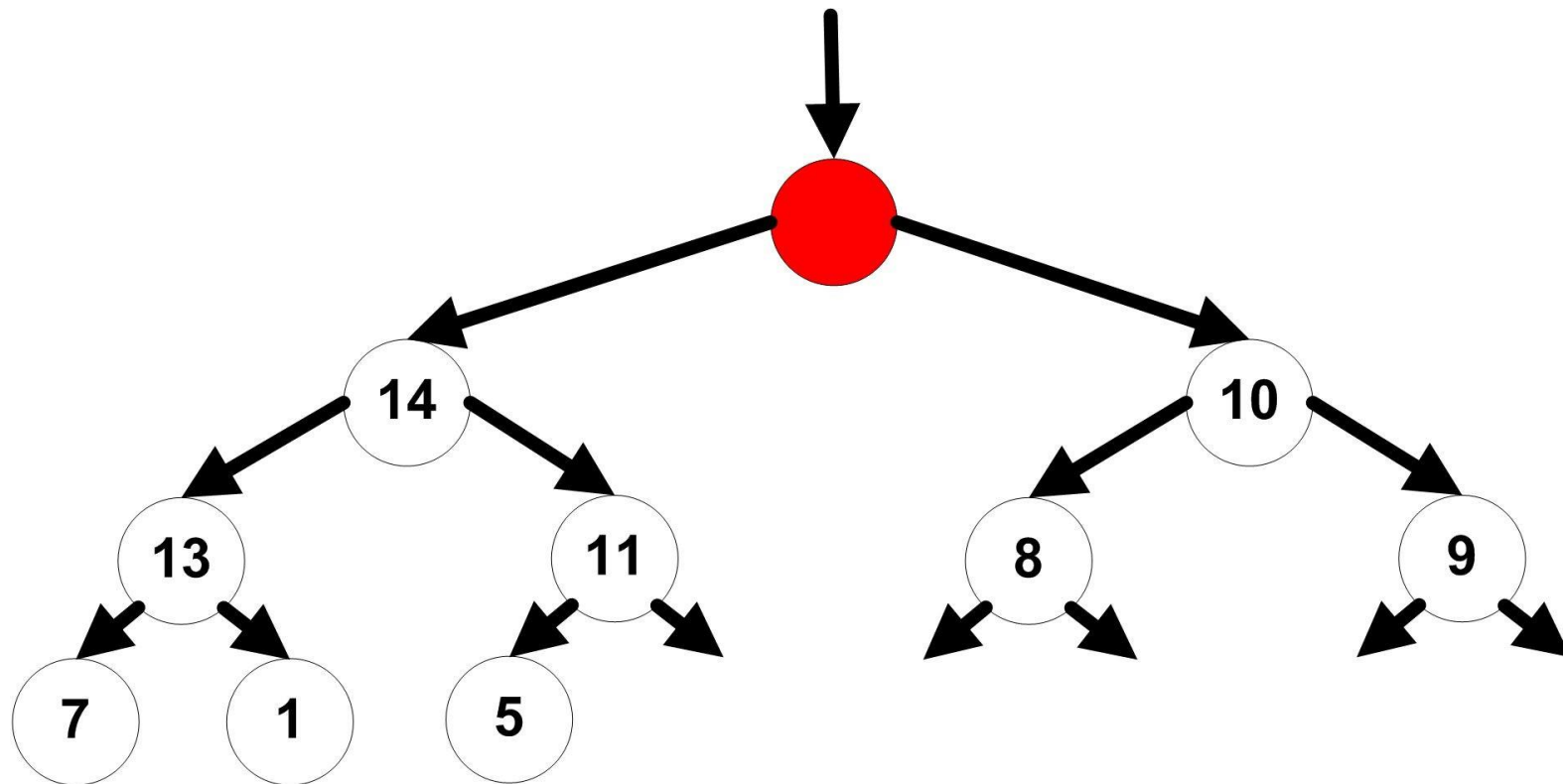
Exercício Resolvido

- Remova a raiz do heap abaixo



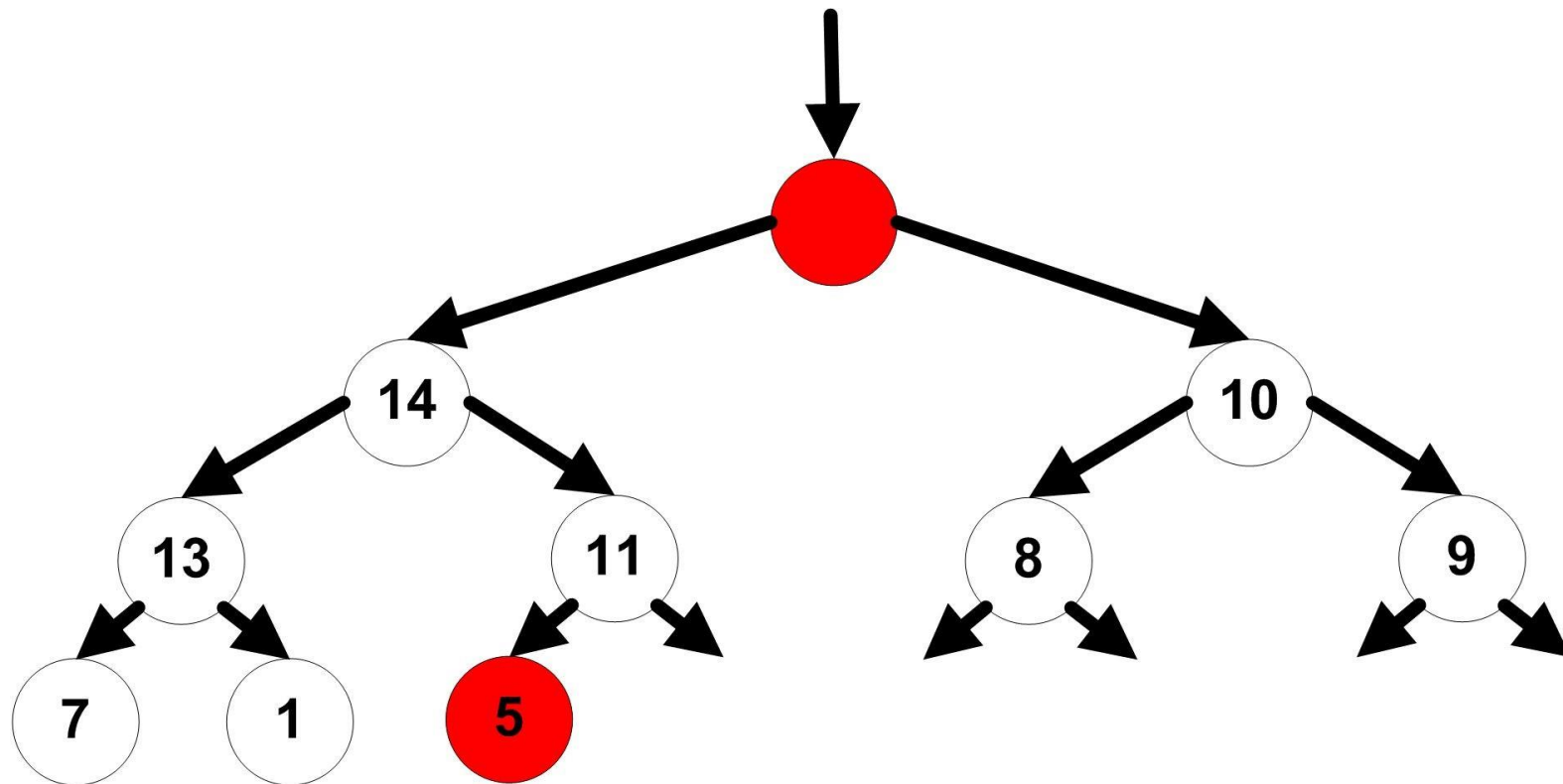
Exercício Resolvido

- Remova a raiz do heap abaixo



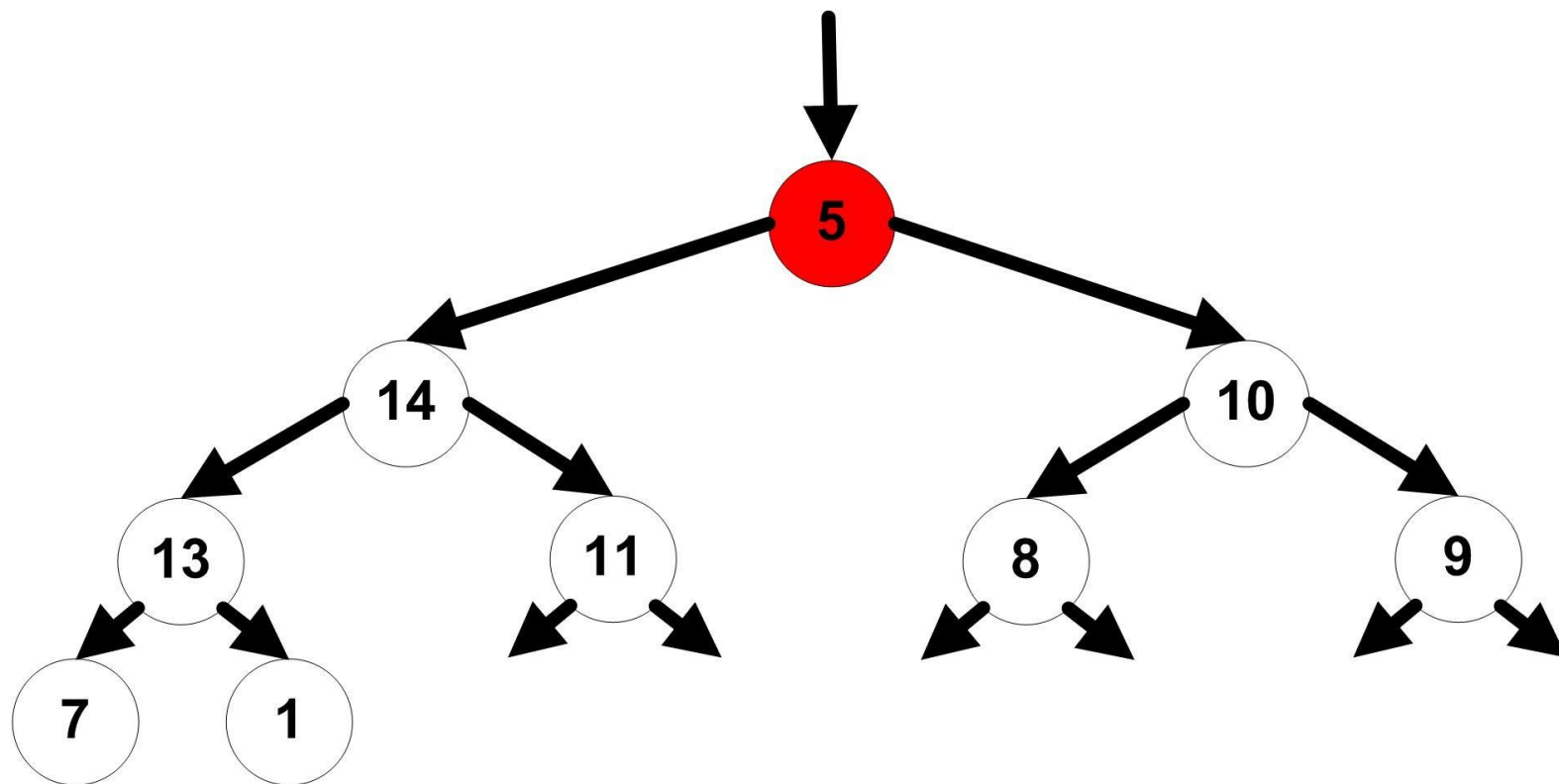
Exercício Resolvido

- Remova a raiz do heap abaixo



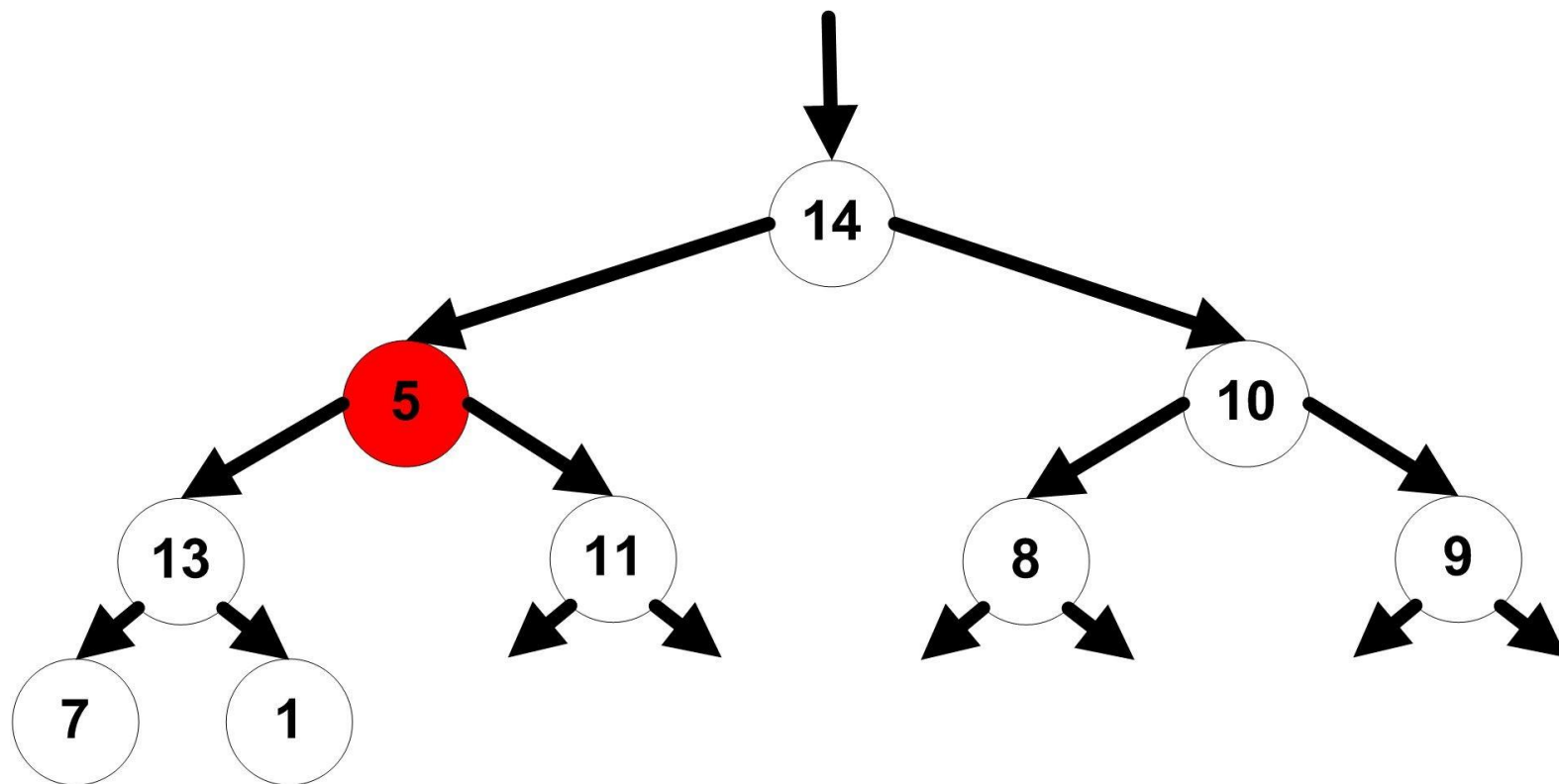
Exercício Resolvido

- Remova a raiz do heap abaixo



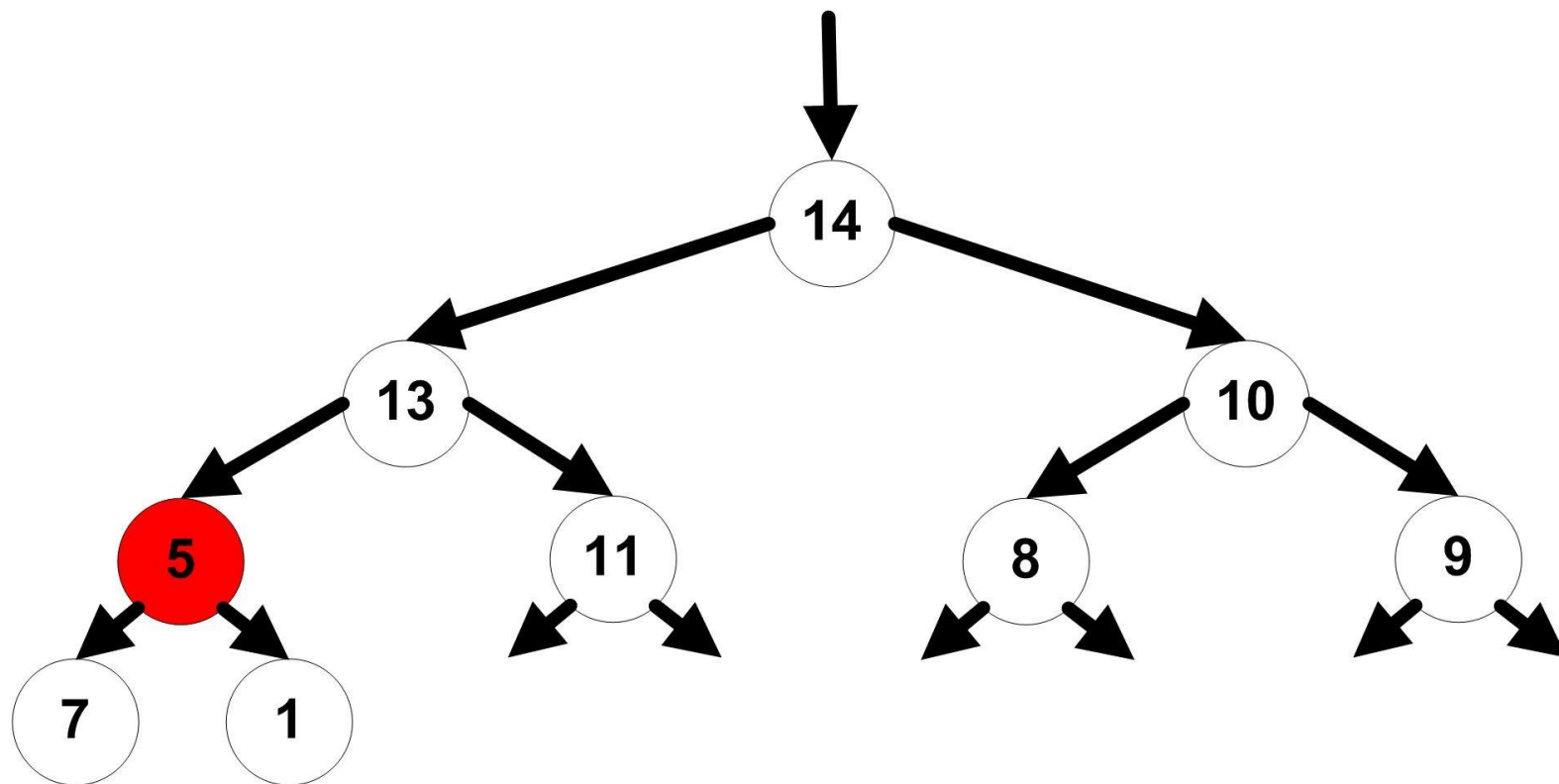
Exercício Resolvido

- Remova a raiz do heap abaixo



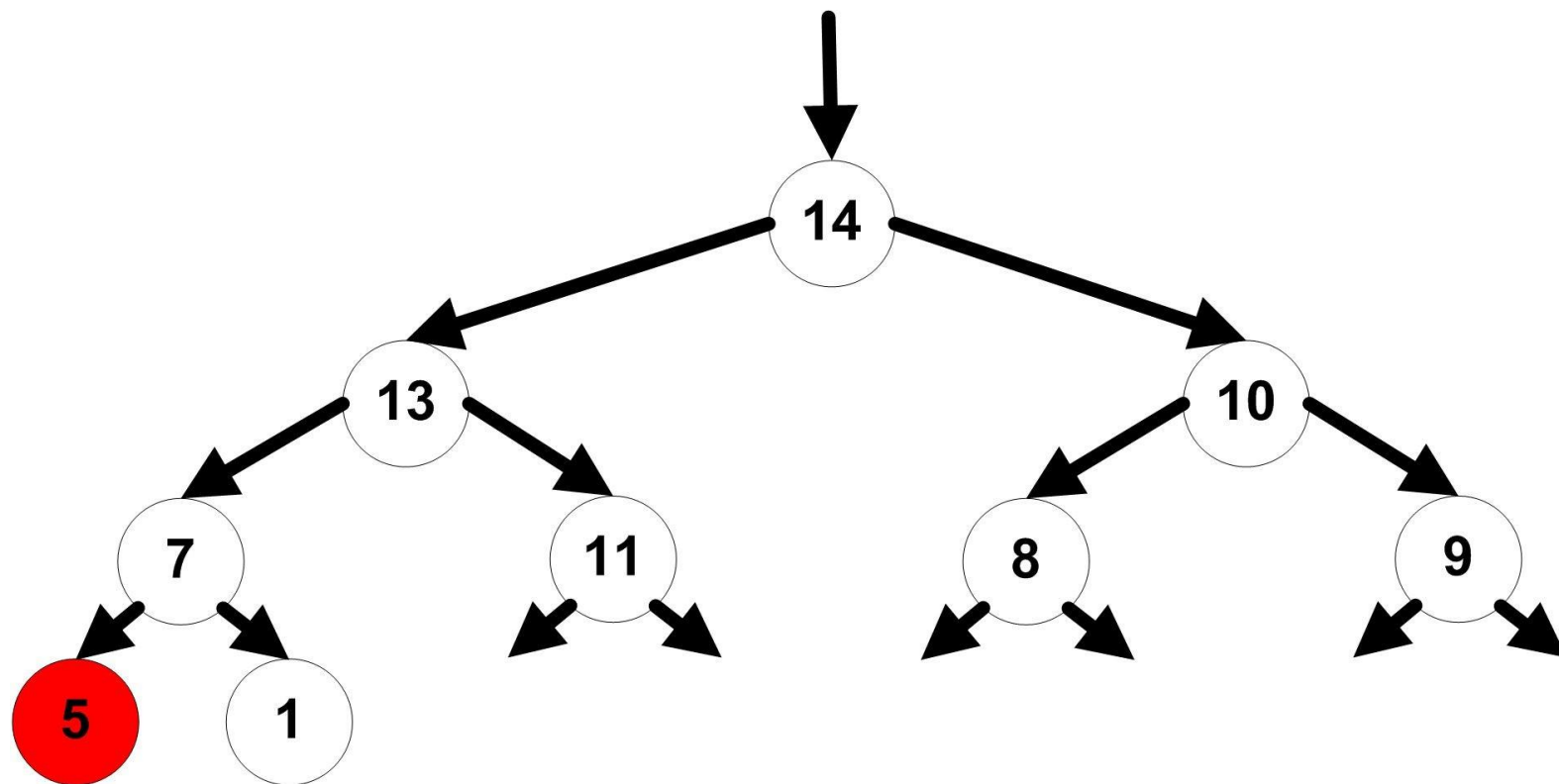
Exercício Resolvido


- Remova a raiz do heap abaixo



Exercício Resolvido

- Remova a raiz do heap abaixo



- Definição de Heap
- Funcionamento básico
- **Algoritmo** 
- Análise do número de comparações e movimentações

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    //Construção do heap  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
  
    //Ordenacao propriamente dita  
    tam = n;  
    while (tam > 1){  
        swap(1, tam--);  
        Reconstruir(array,tam);  
    }  
}
```

```
void Construir(int[] array, int tam){  
    ■ ■ ■  
}
```

```
void Reconstruir(int[] array, int tam)  
    ■
```

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    . . .
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {
```

```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }
```

```
        ■ ■ ■
```

```
void Construir(int[] array, int tam){
```

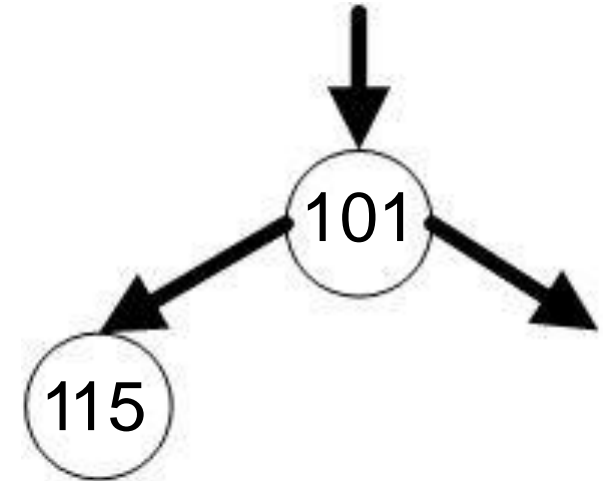
```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



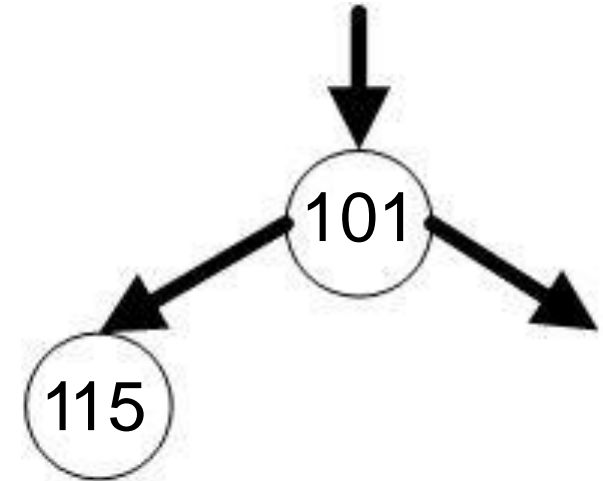
101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) { true: 2 <= 6

    int tam;
    for (tam = 2; tam <= n; tam++){
        Construir(array,tam);
    }
    ...
}
```

```
void Construir(int[] array, int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

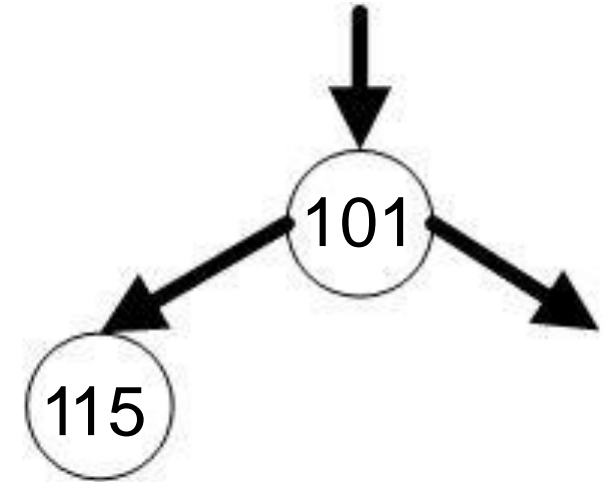


101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

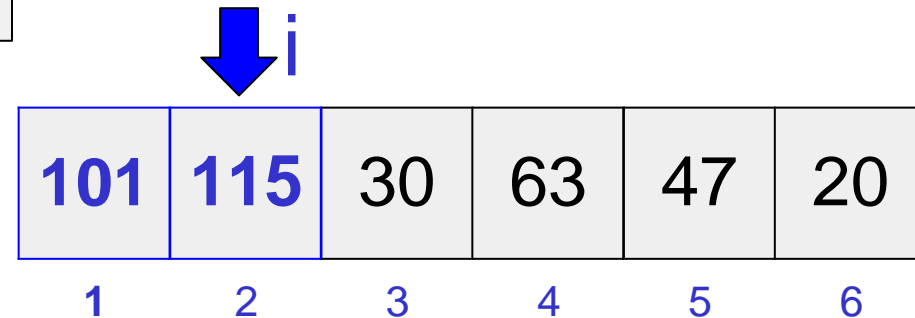
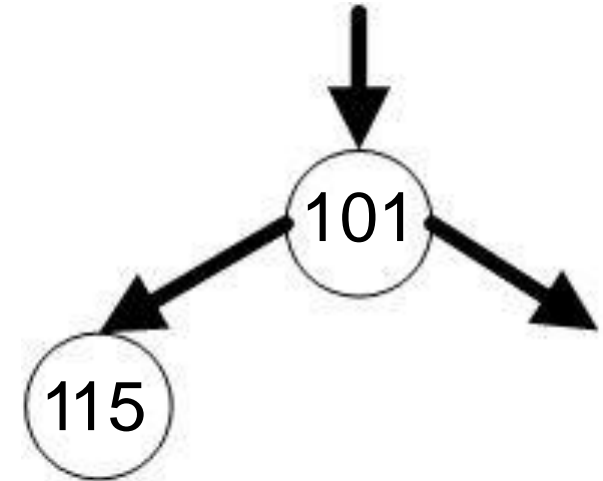


101	115	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

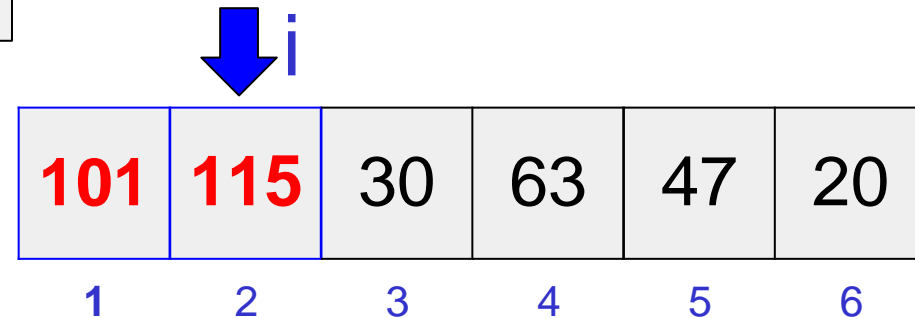
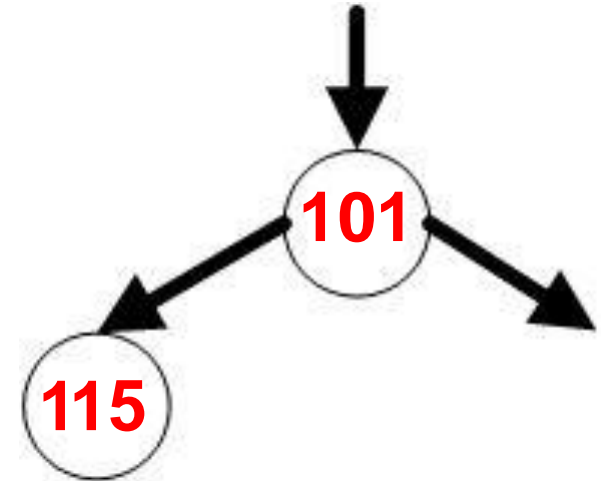
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

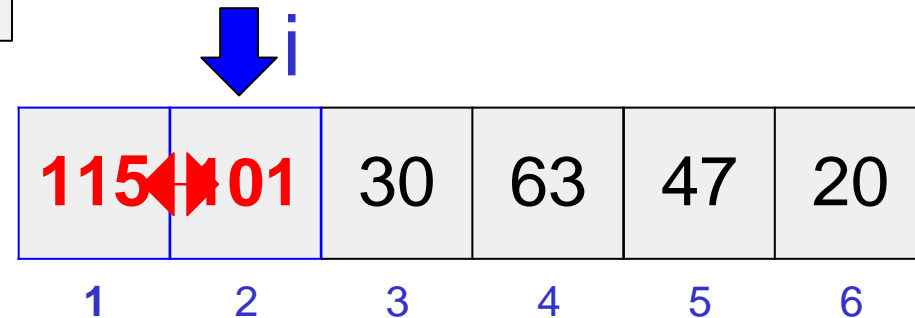
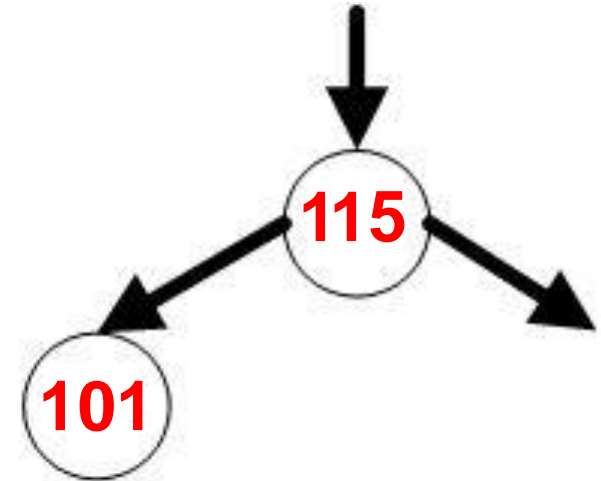
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
    true: 2 > 1 && 115 > 101  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

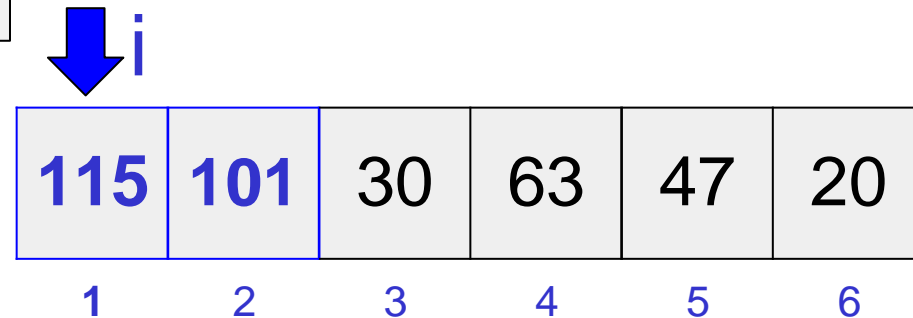
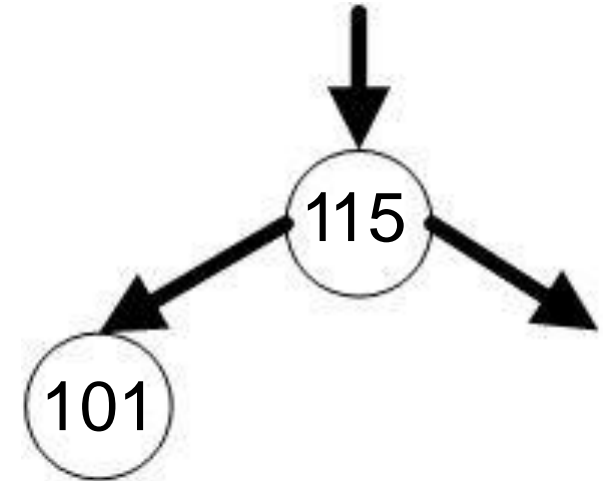
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {
```

```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++){
```

```
        Construir(array,tam);
```

```
    }
```

```
        ...
```

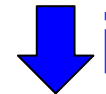
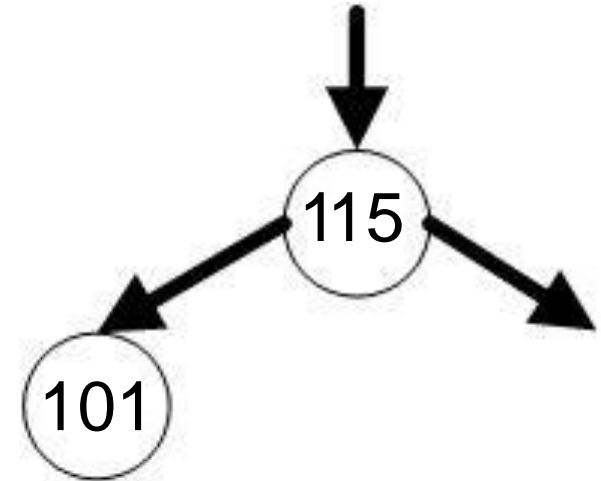
```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
    } // false: 1 > 1 && ...
```

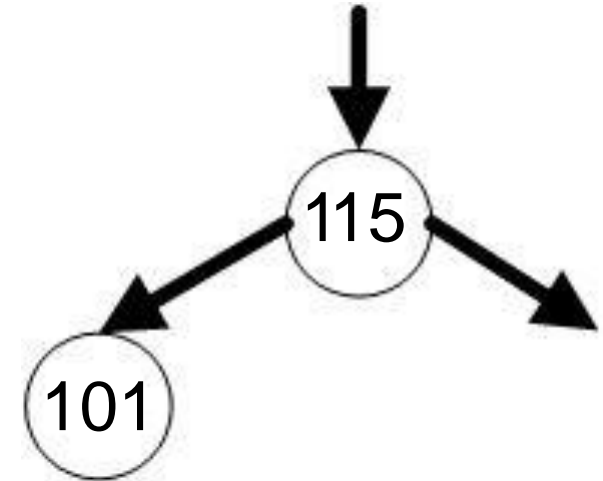


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {
```

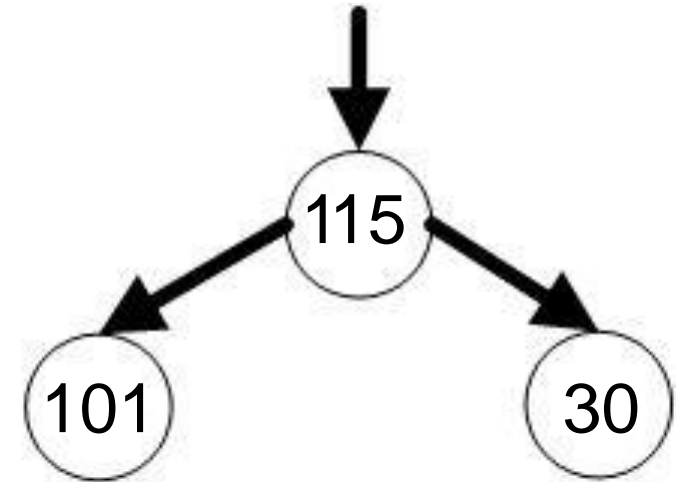
```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++) {
        Construir(array, tam);
    }
```

```
    ...
```

```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {
```

```
    int tam;
```

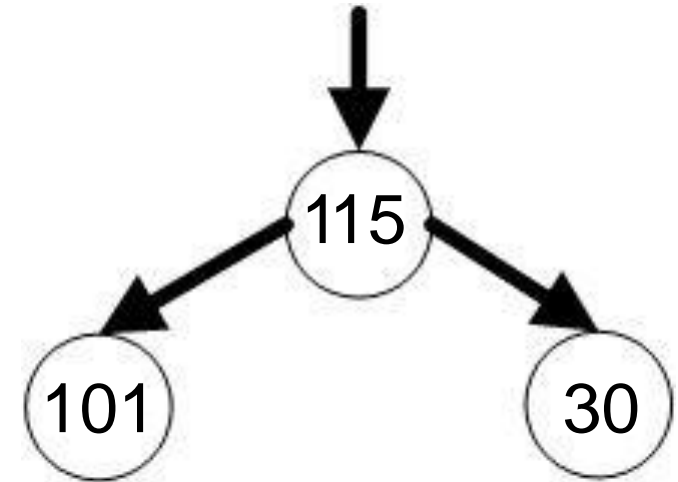
true: $3 \leq 6$

```
    for (tam = 2; tam <= n; tam++){
        Construir(array,tam);
    }
```

...

```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

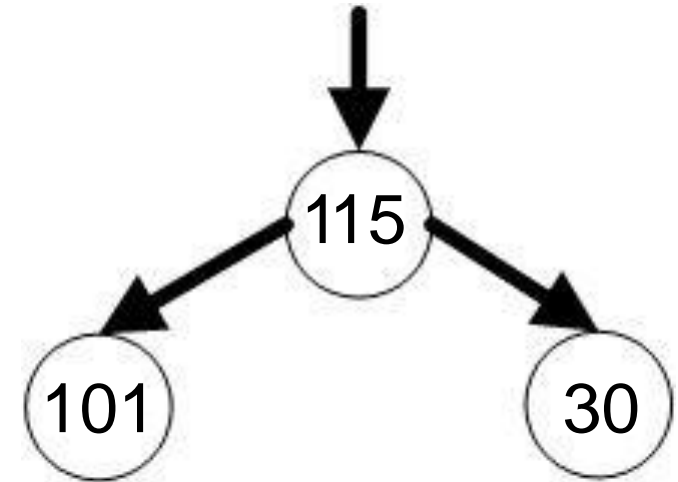


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

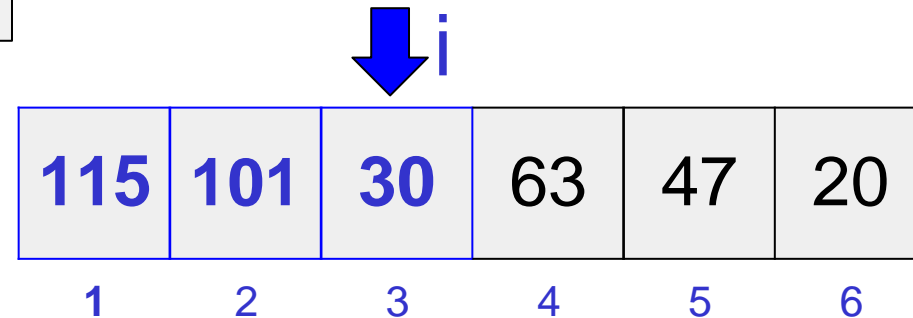
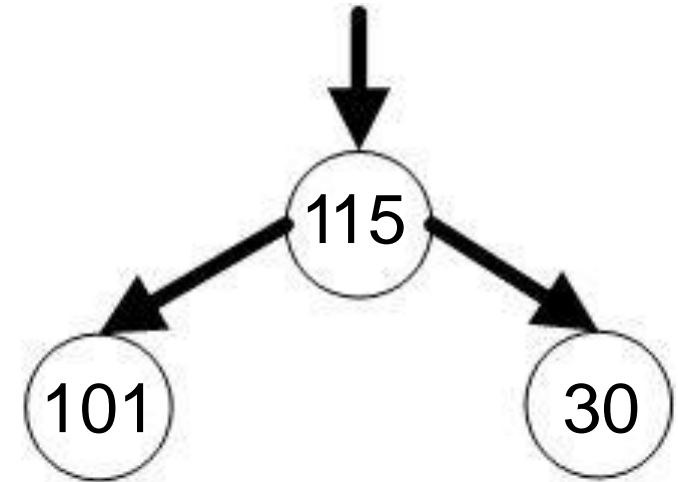


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

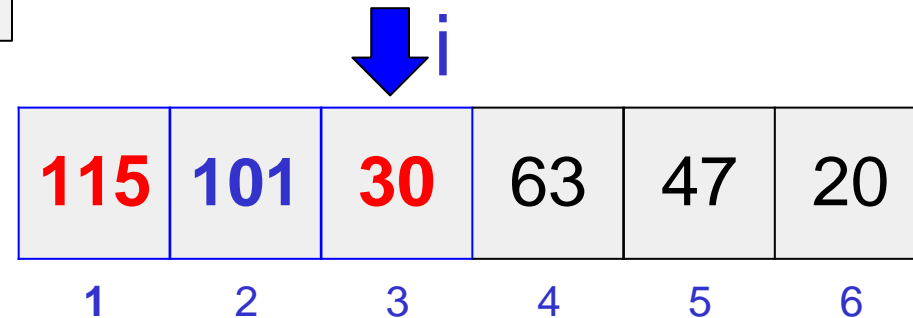
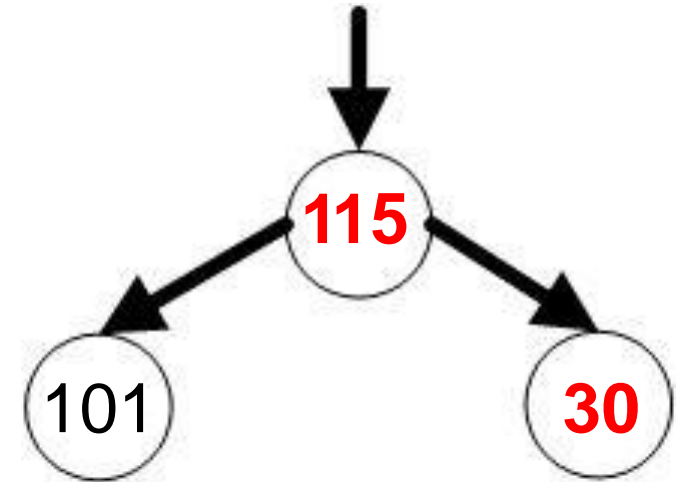
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

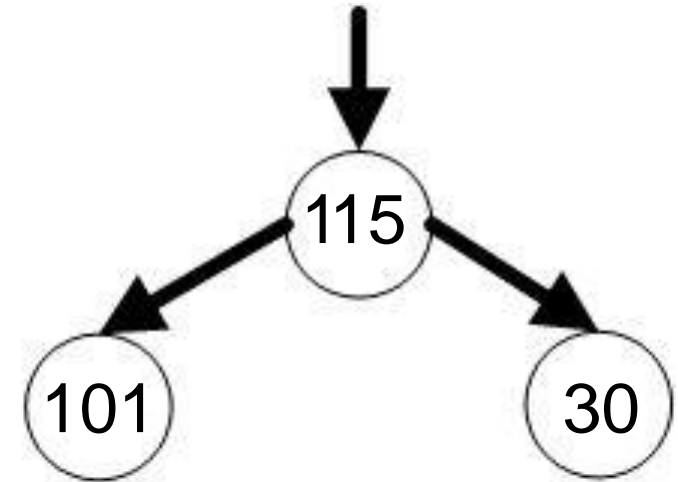
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
    false: 3 > 1 && 30 > 115  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {
```

```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++) {
```

```
        Construir(array, tam);
```

```
    }
```

```
        . . .
```

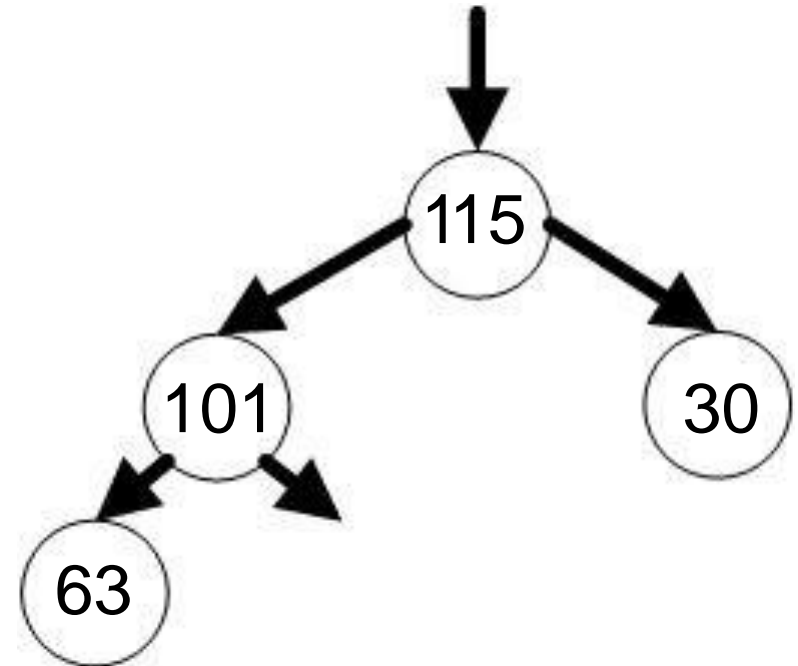
```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```



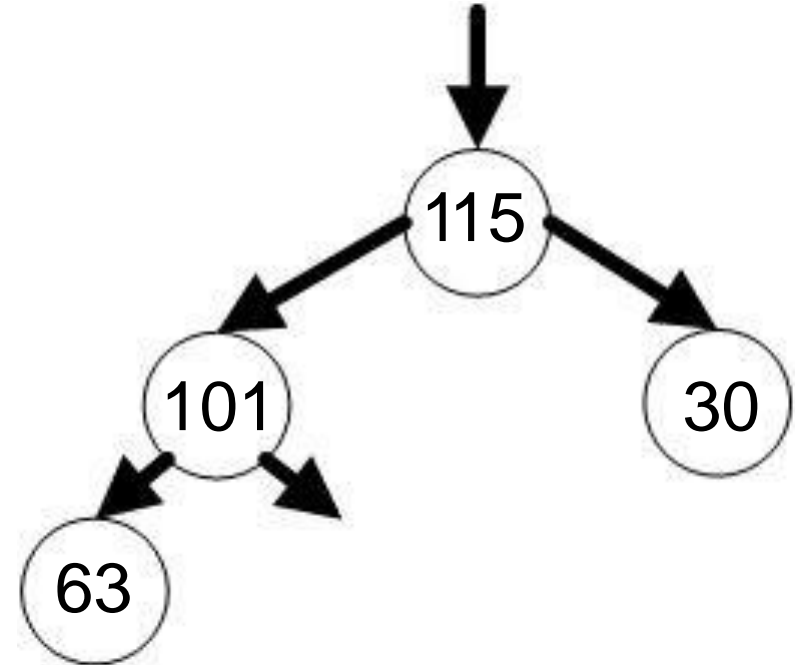
115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

true: $4 \leq 6$

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

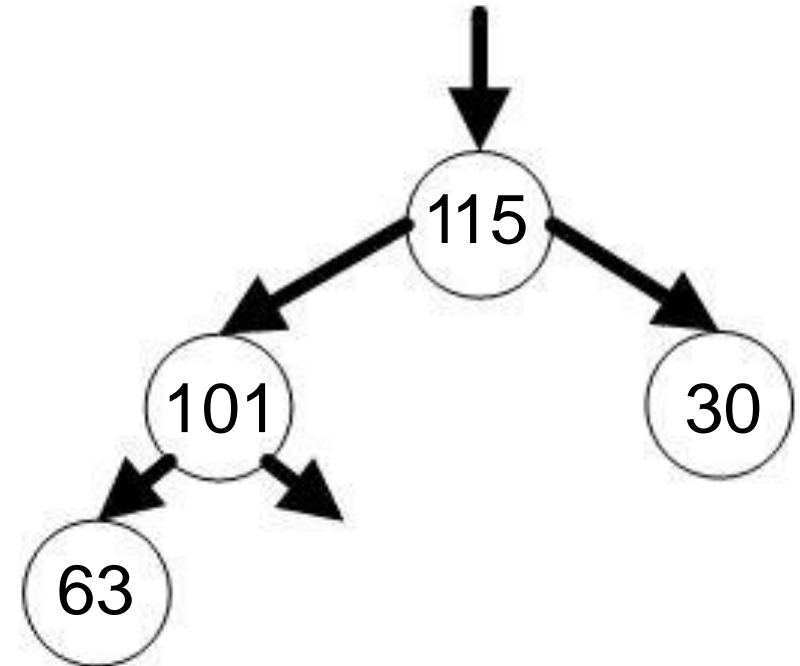


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

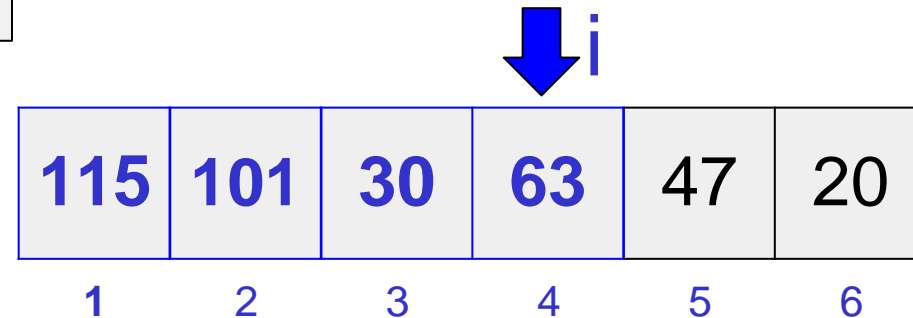
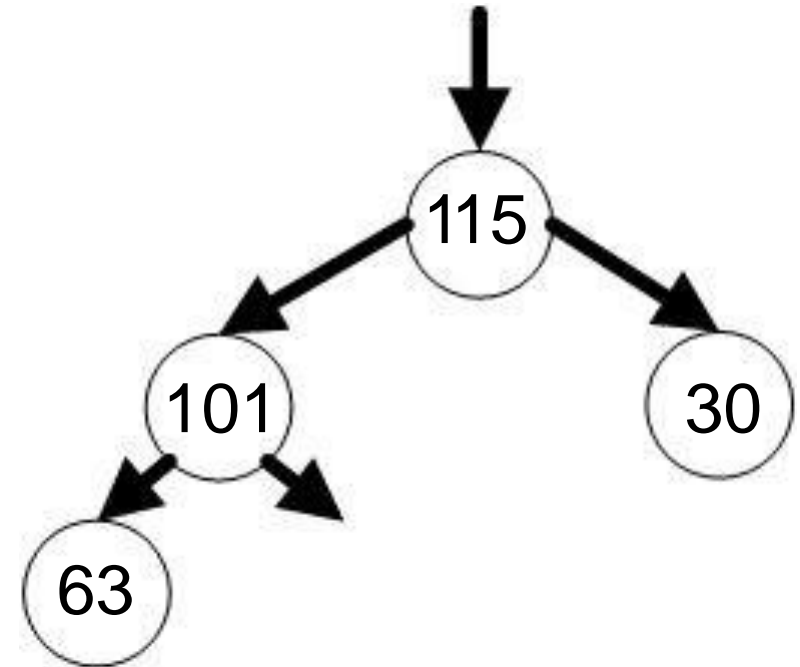


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

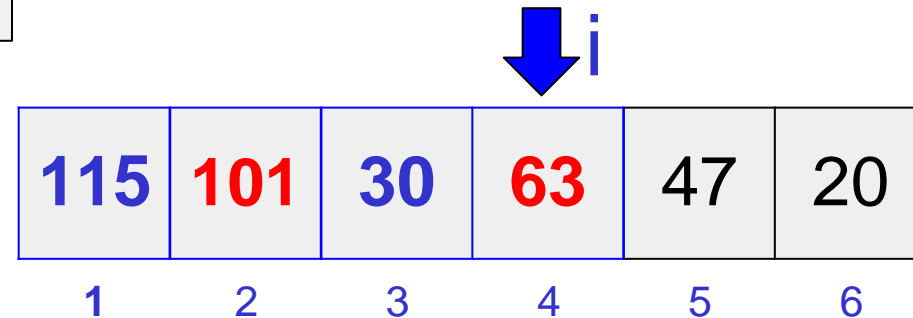
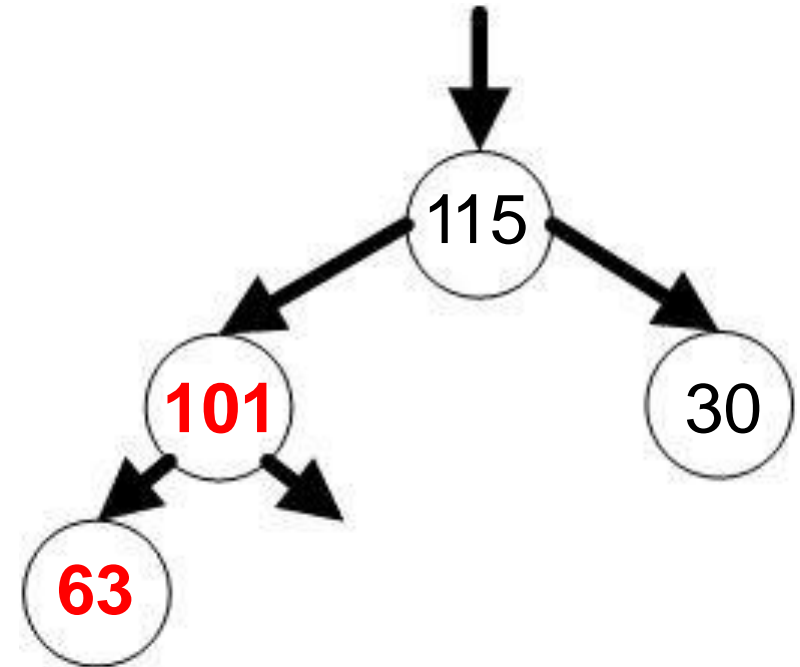
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

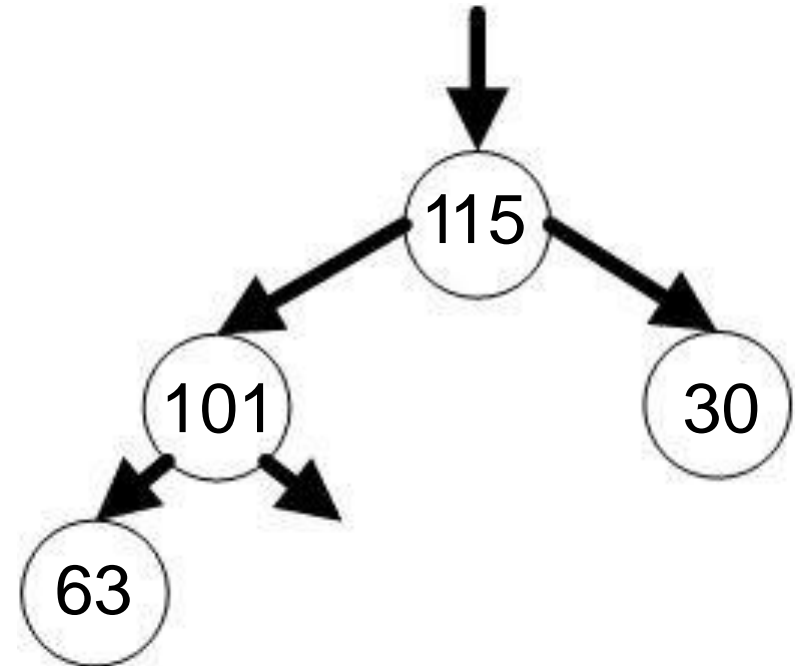
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
    false: 4 > 1 && 63 > 101  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {
```

```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++) {
```

```
        Construir(array, tam);
```

```
    }
```

```
        ...
```

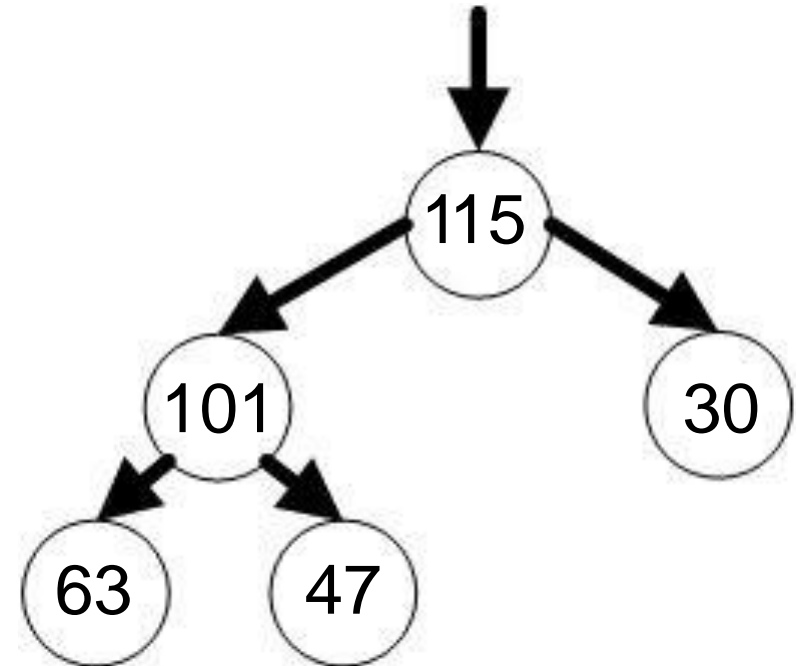
```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {
```

true: $5 \leq 6$

```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++){
```

```
        Construir(array,tam);
```

```
    }
```

```
        . . .
```

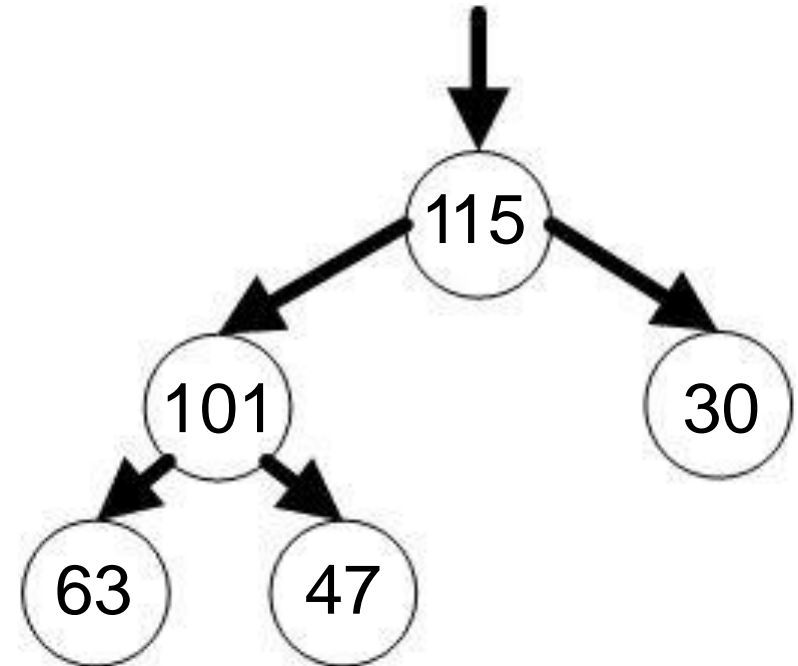
```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```

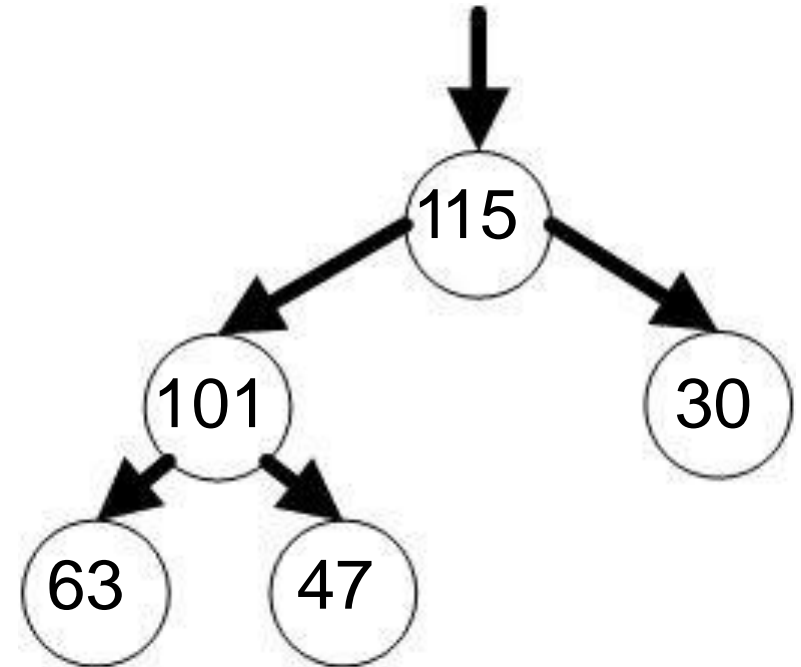


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

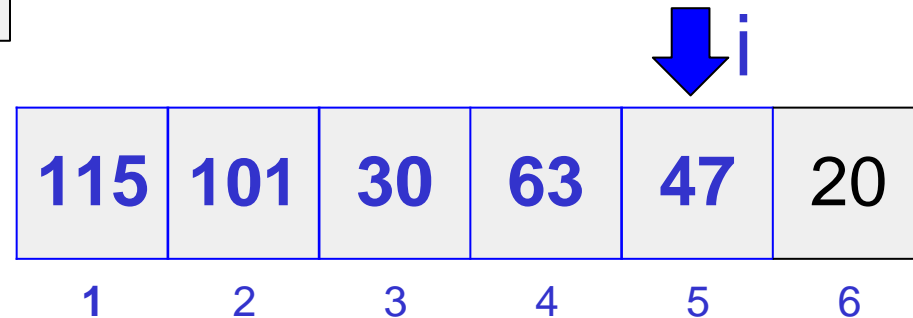
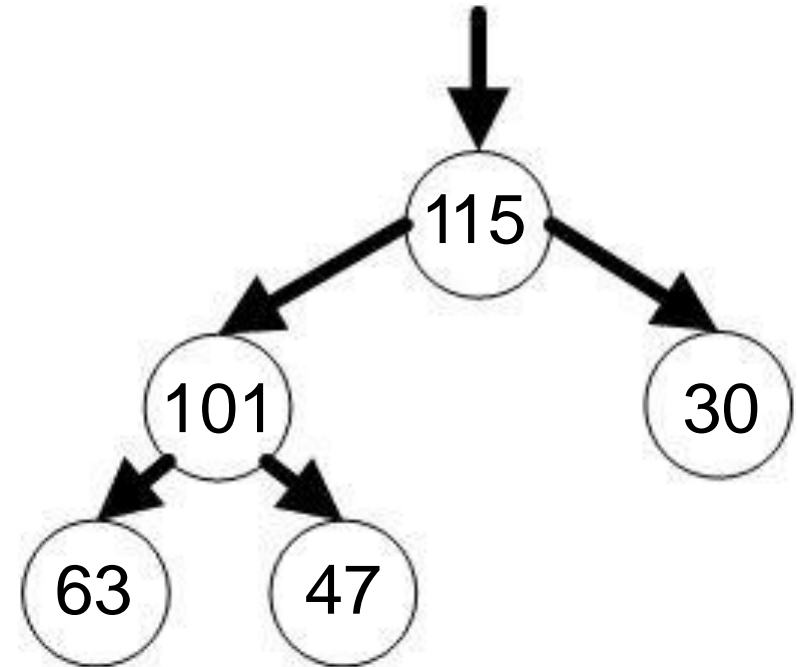


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

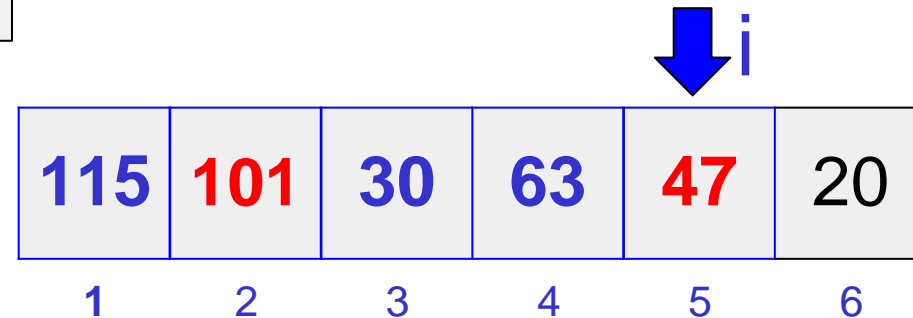
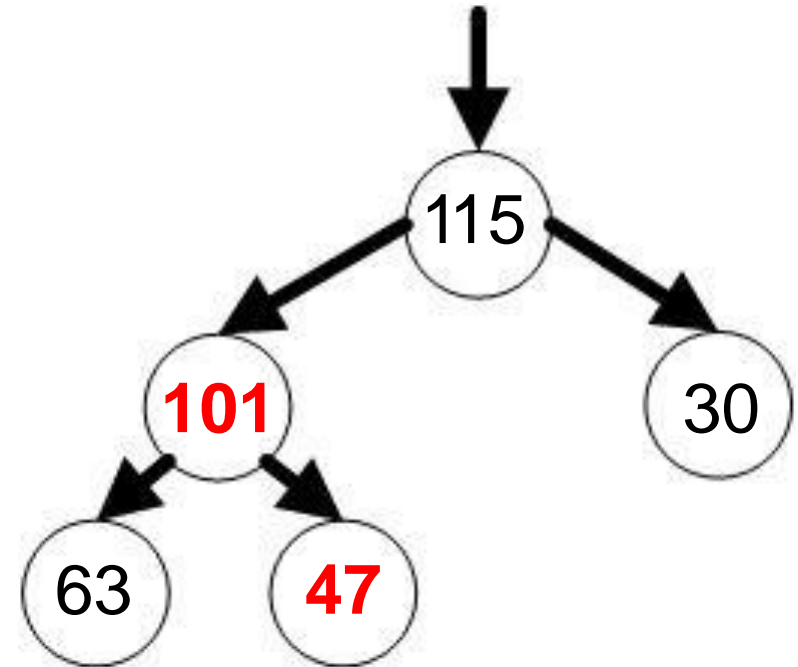
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

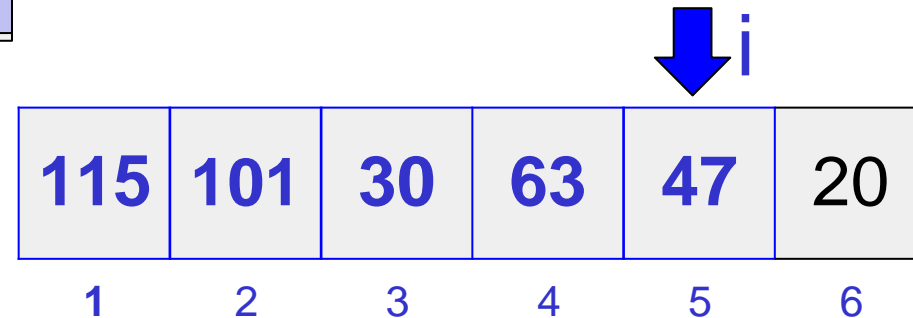
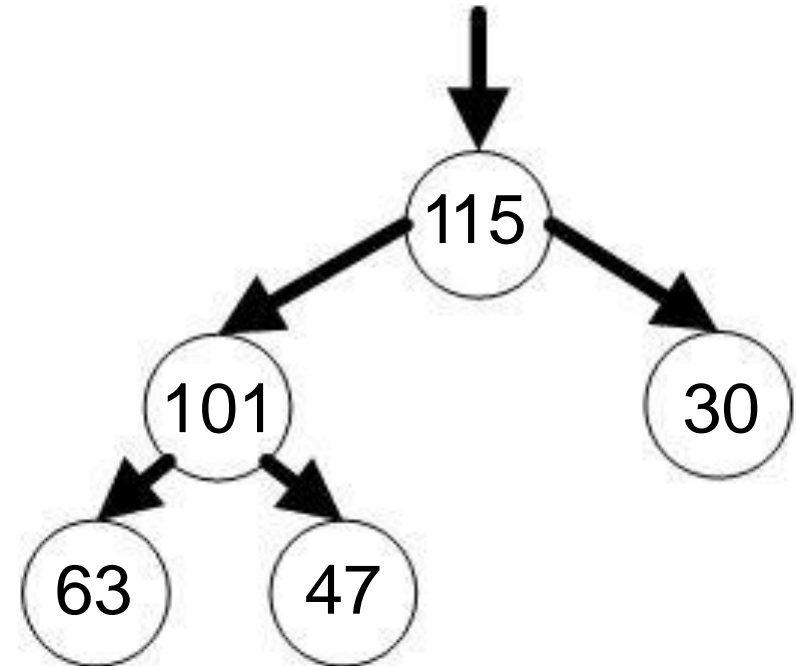
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
    false: 5 > 1 && 47 > 101  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {
```

```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++) {
```

```
        Construir(array, tam);
```

```
    }
```

```
        ...
```

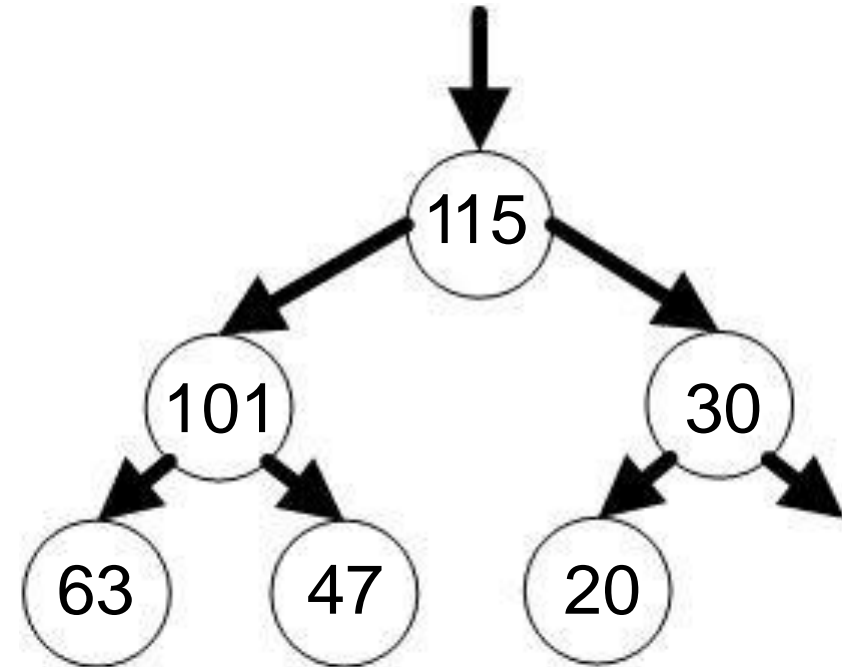
```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) { true: 6 <= 6
```

```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++){
```

```
        Construir(array,tam);
```

```
    }
```

```
        . . .
```

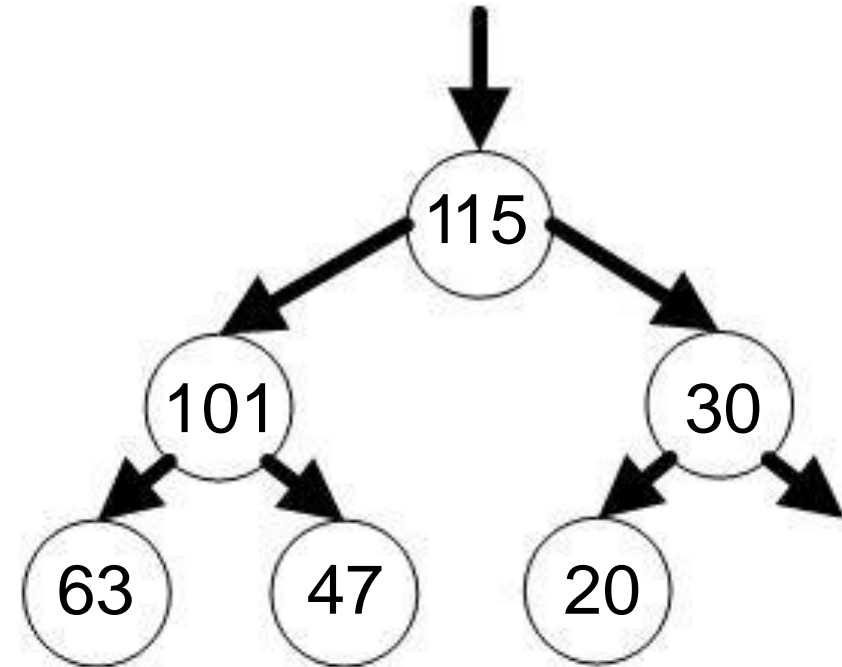
```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```

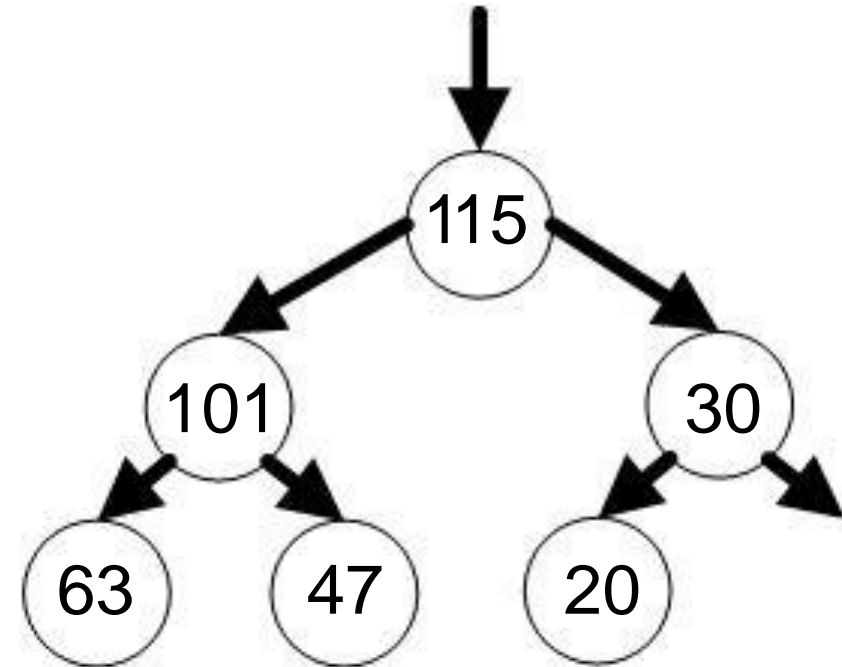


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

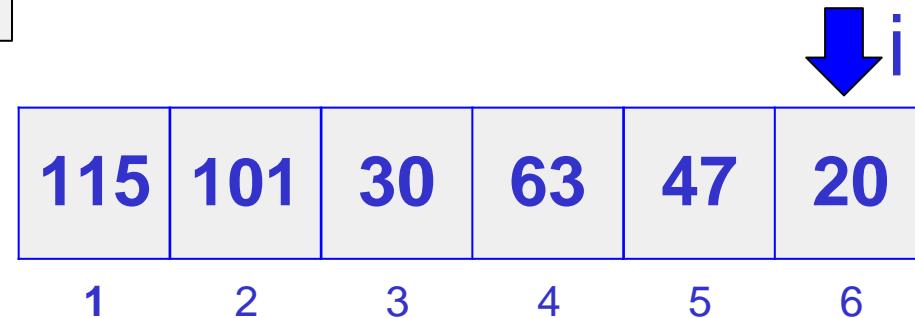
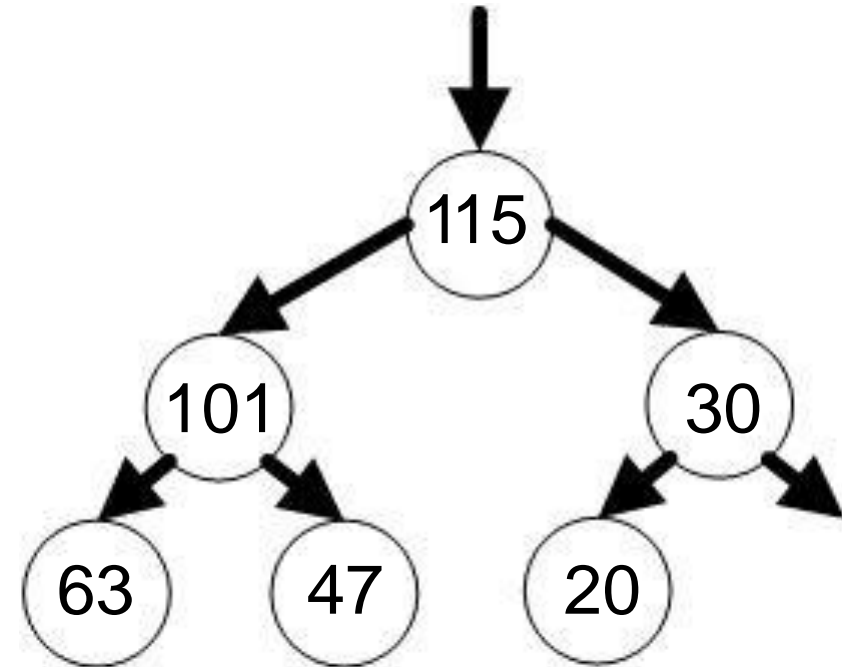


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

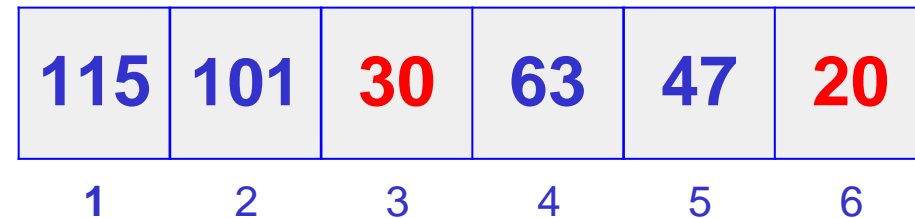
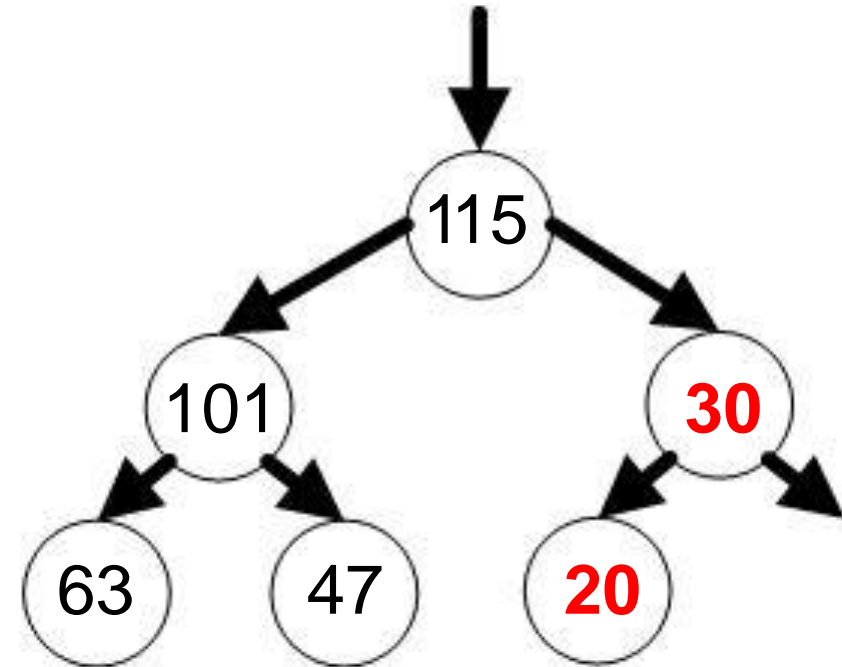
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

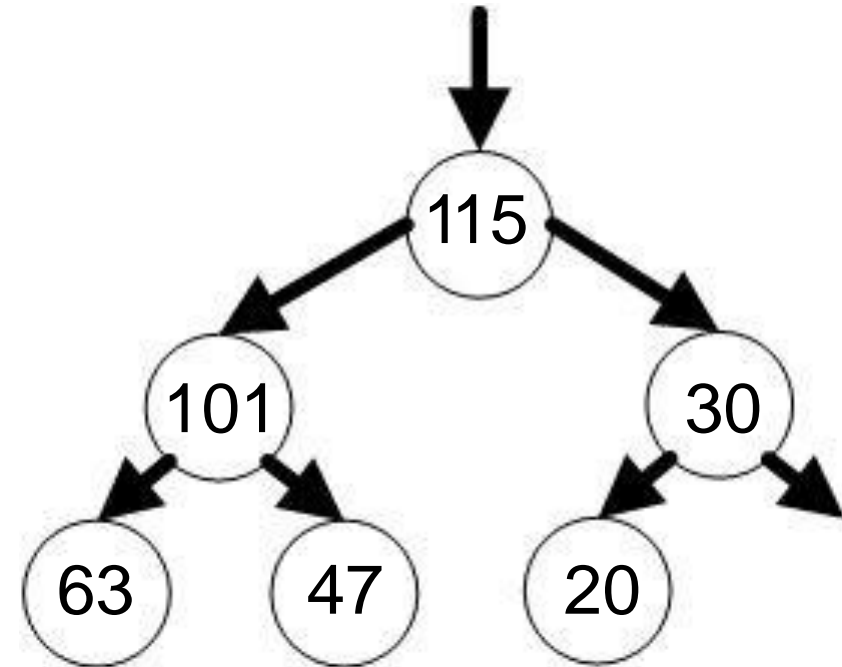
```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
    false: 6 > 1 && 20 > 30  
}
```



Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
    ...  
}
```

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {
```

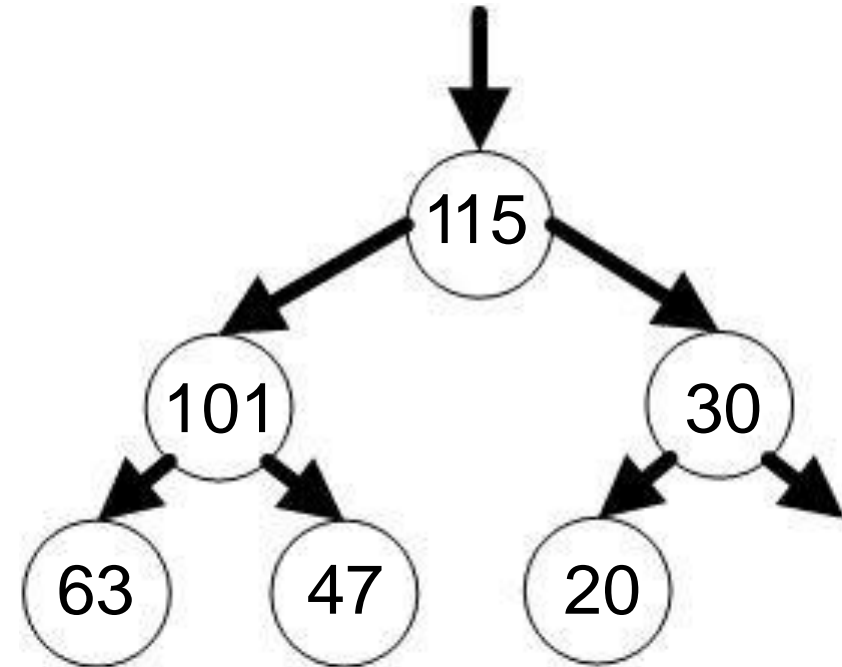
```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++) {
        Construir(array, tam);
    }
```

```
    ...
```

```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) { false: 7 <= 6
```

```
    int tam;
```

```
    for (tam = 2; tam <= n; tam++){
```

```
        Construir(array,tam);
```

```
    }
```

```
        . . .
```

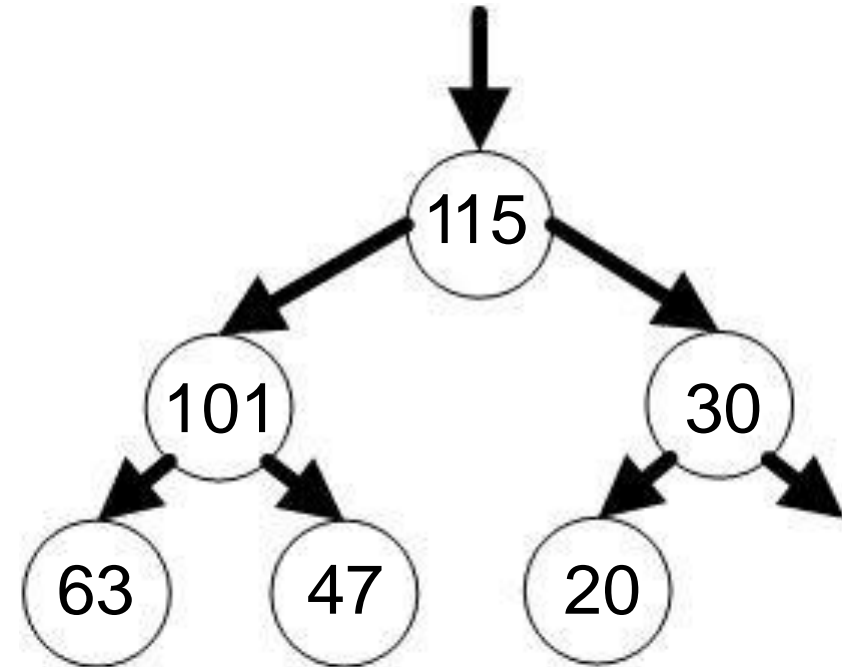
```
void Construir(int[] array, int tam){
```

```
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
```

```
        swap(i, i/2);
```

```
    }
```

```
}
```



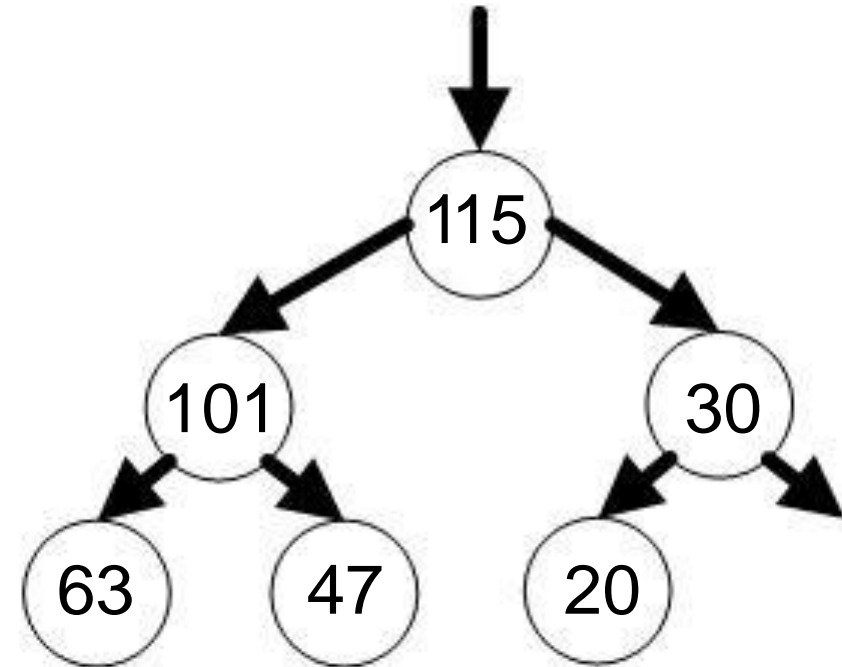
115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam)  
    }  
}
```

■ ■ ■

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

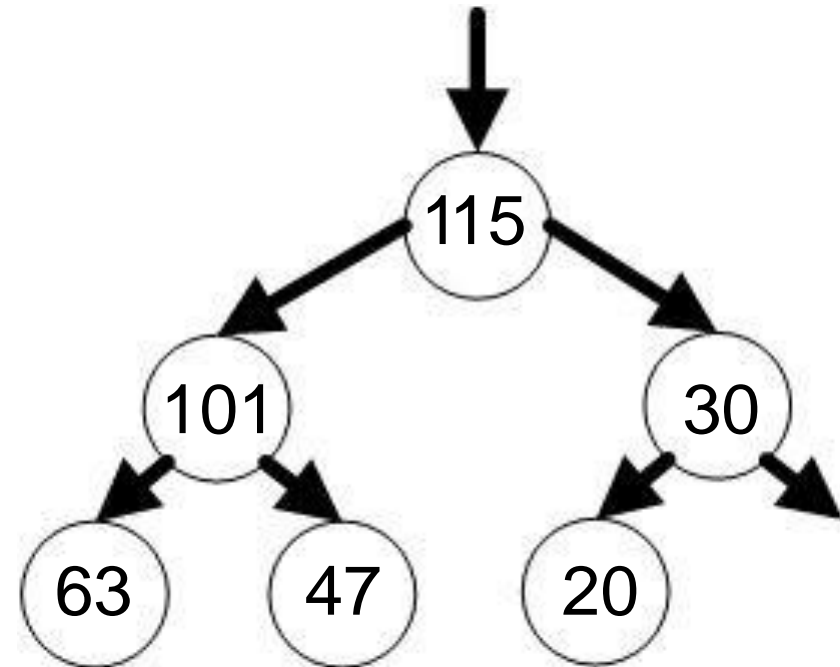


115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
}
```

■ ■ ■



115	101	30	63	47	20
1	2	3	4	5	6

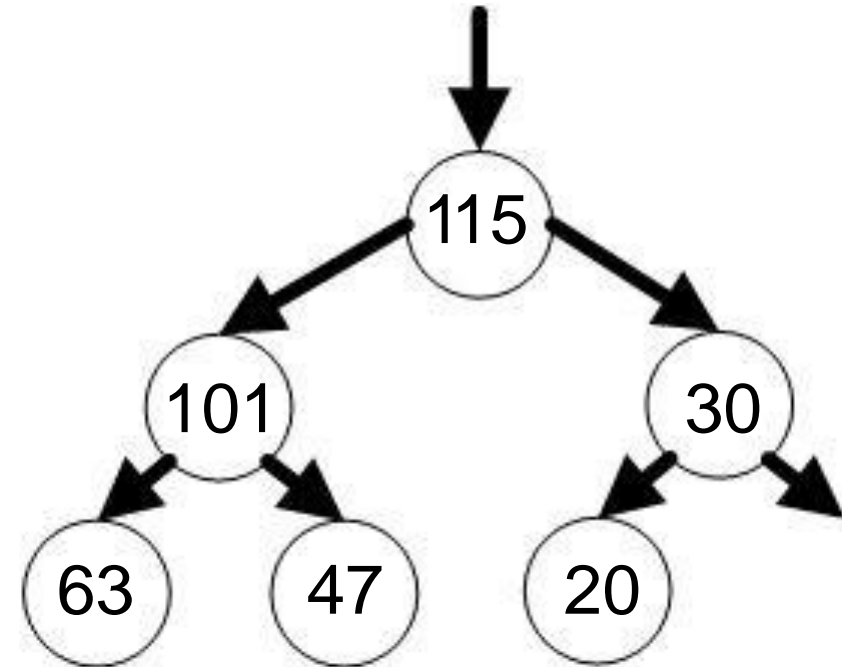
Algoritmo

```

void Heapsort(int[] array, int n) {

    int tam;
    for (tam = 2; tam <= n; tam++){
        Construir(array,tam);
    }

    //Ordenacao propriamente dita
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }
}
    
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

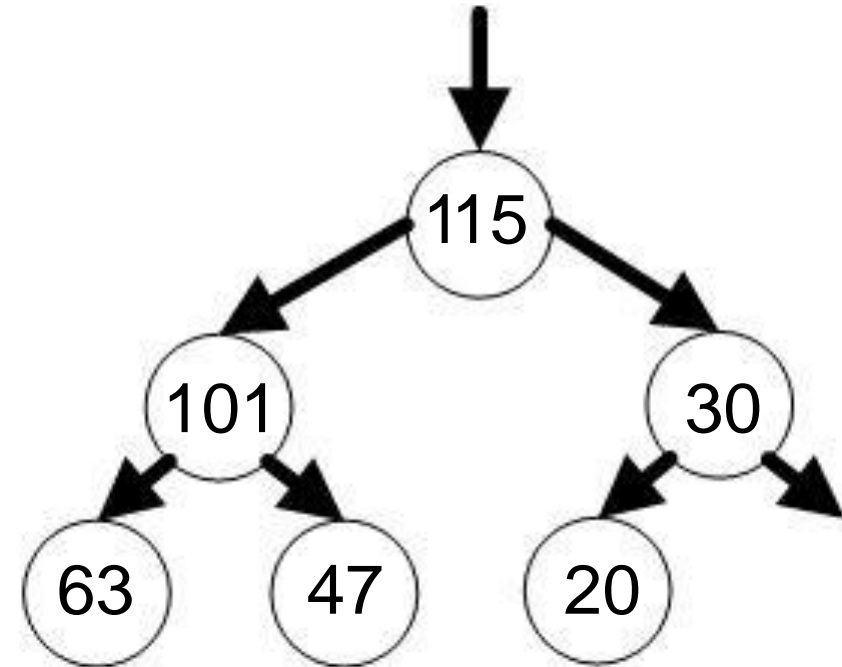
```

void Heapsort(int[] array, int n) {

    int tam;
    for (tam = 2; tam <= n; tam++){
        Construir(array,tam);
    }

    //Ordenacao propriamente dita

    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }
}
    
```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```

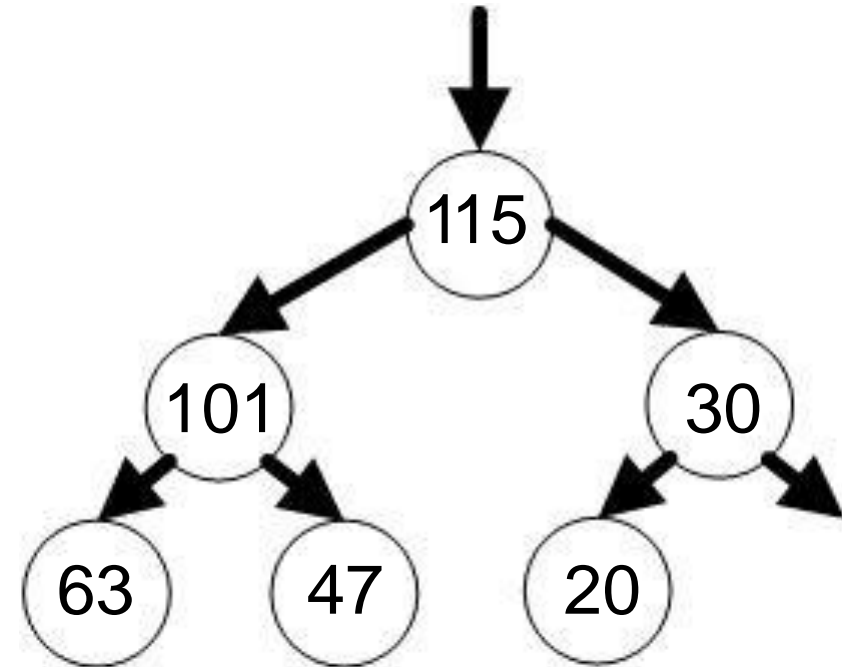
tam = n;
while (tam > 1){
    swap(1, tam--);
    Reconstruir(array, tam);
}

```

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;

```

tam

6

```

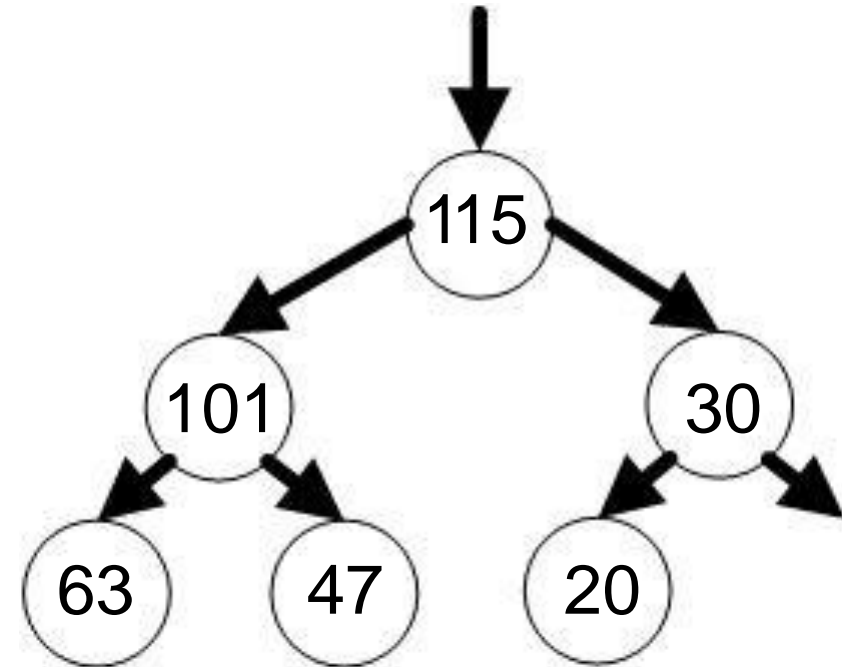
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }
}

```

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



115

101

30

63

47

20

1

2

3

4

5

6

Algoritmo

```

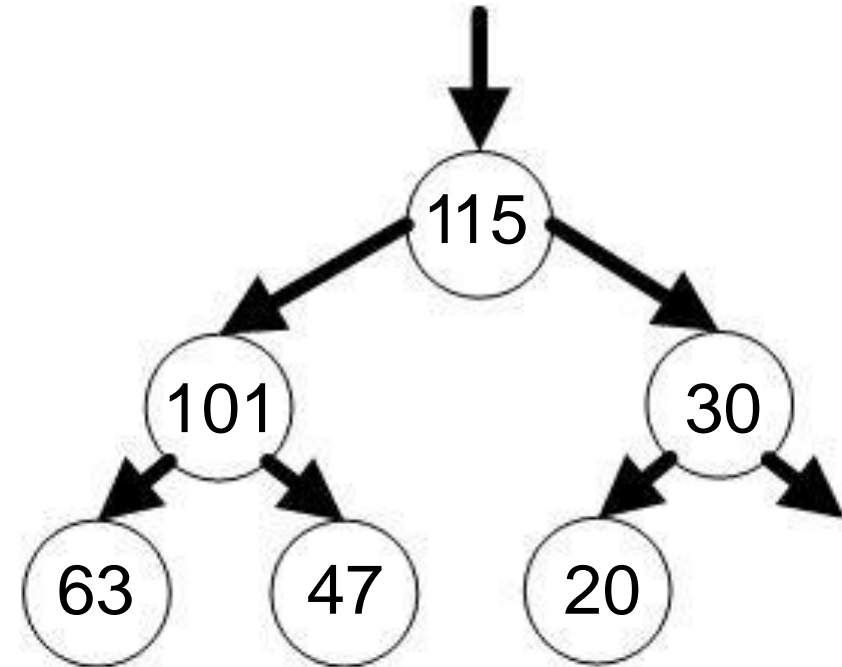
    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }
    }
    true: 6 > 1

```

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



115	101	30	63	47	20
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

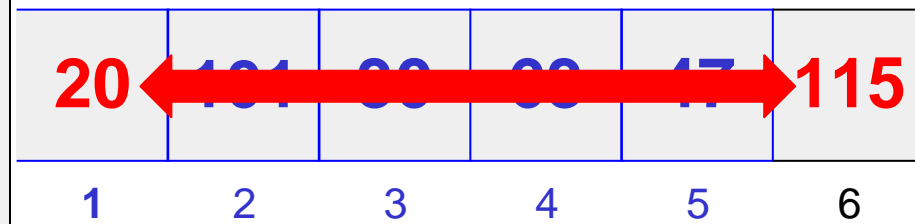
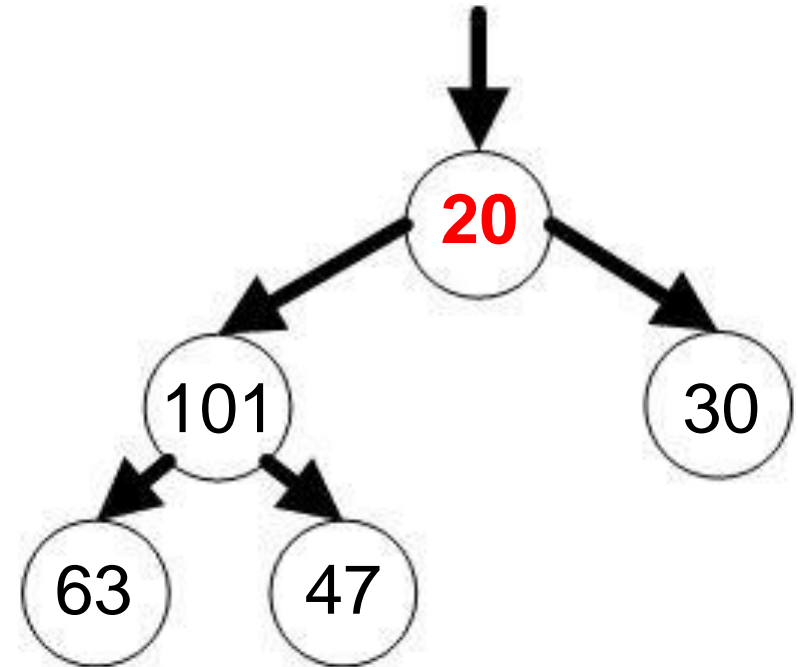
```

tam **5**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

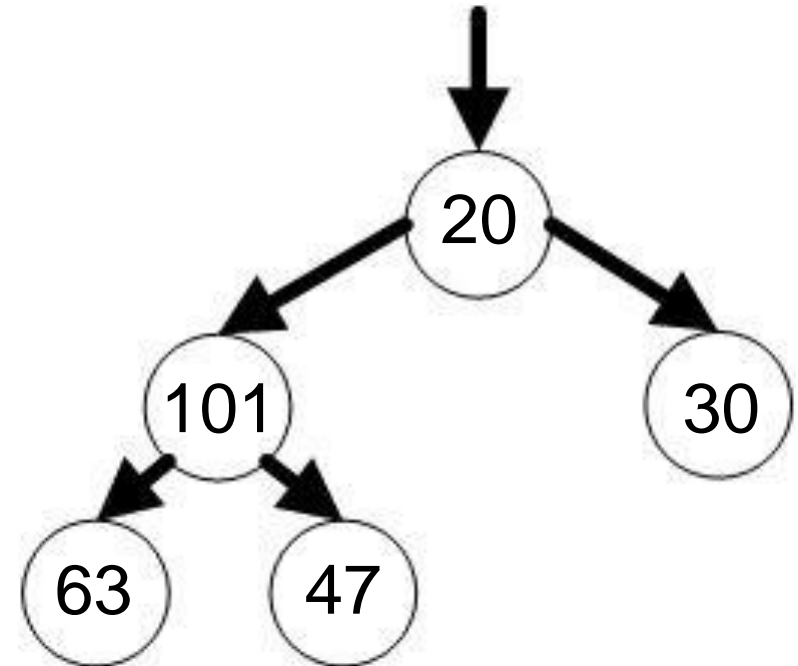
```

tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



20	101	30	63	47	115
1	2	3	4	5	6

Algoritmo

...

```
tam = n;
while (tam > 1){
    swap(1, tam--);
```

tam

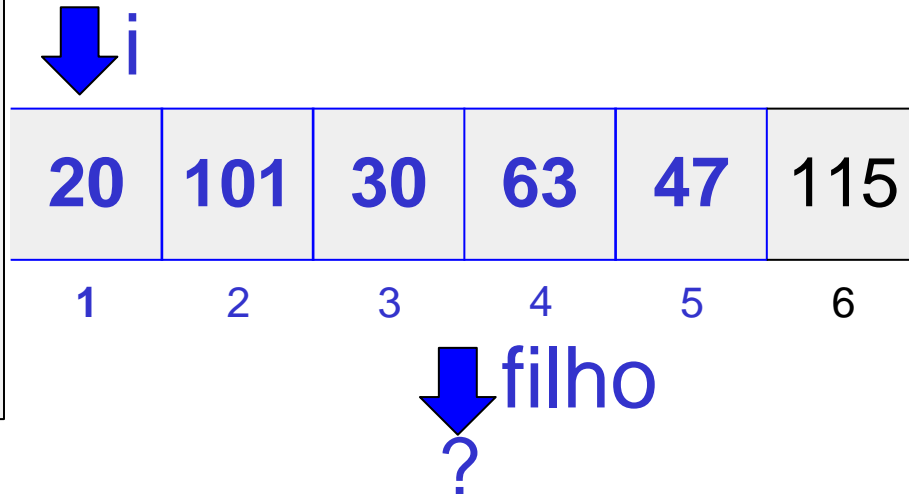
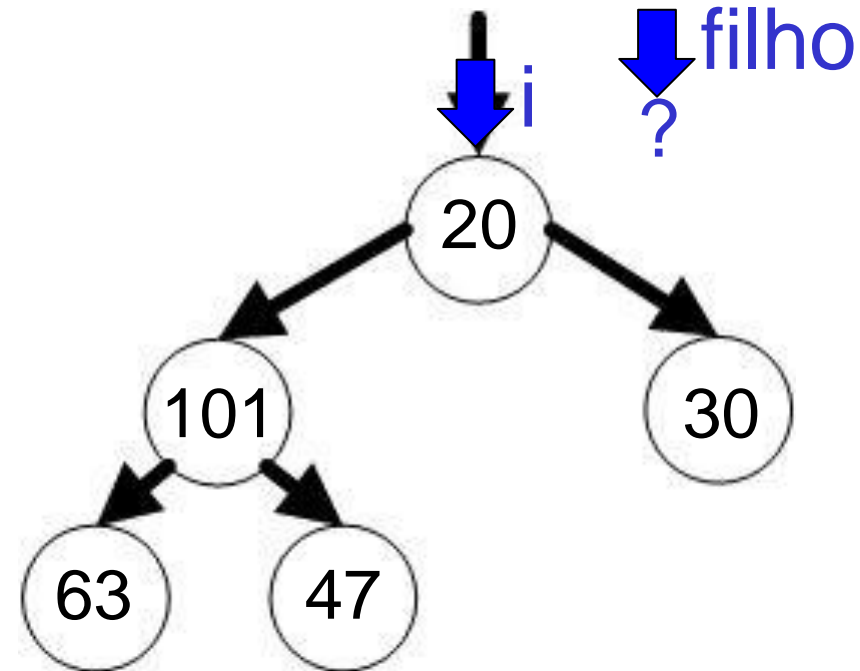
5

```
Reconstruir(array,tam);
}
```

```
void Reconstruir(int[] array, int tam){
```

```
int i = 1;
```

```
while (HasFilho(i, tam) == true){
    int filho = GetMaiorFilho(i, tam);
    if (array[i] < array[filho]) {
        swap(i, filho);
        i = filho;
    } else {
        i = tam;
    }
}
```



Algoritmo

```

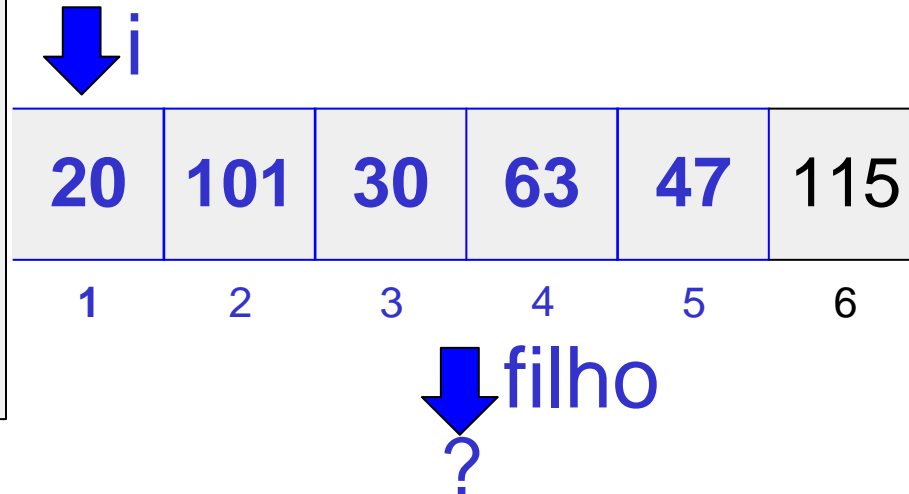
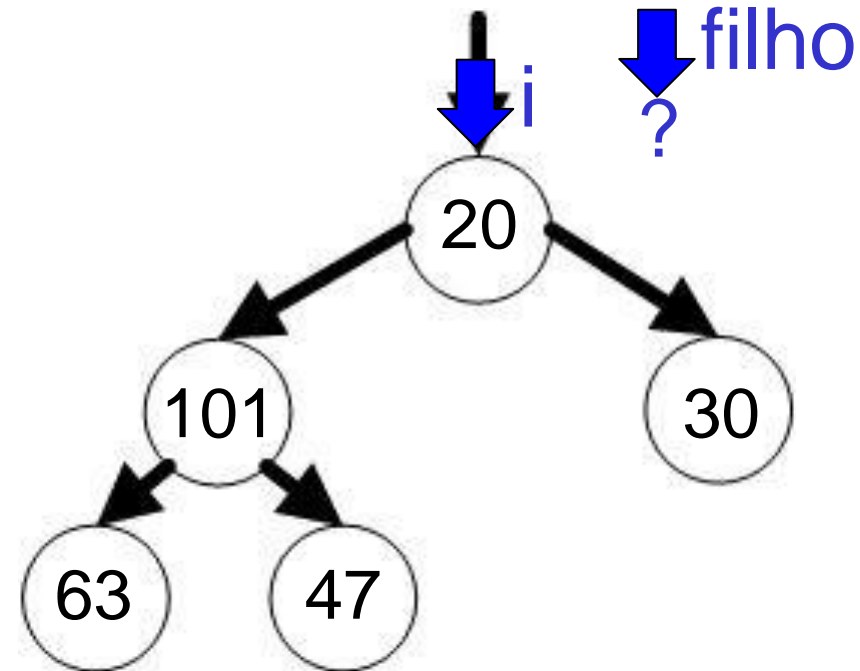
    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }
  
```

tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
  
```

true: 1 <= 2



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

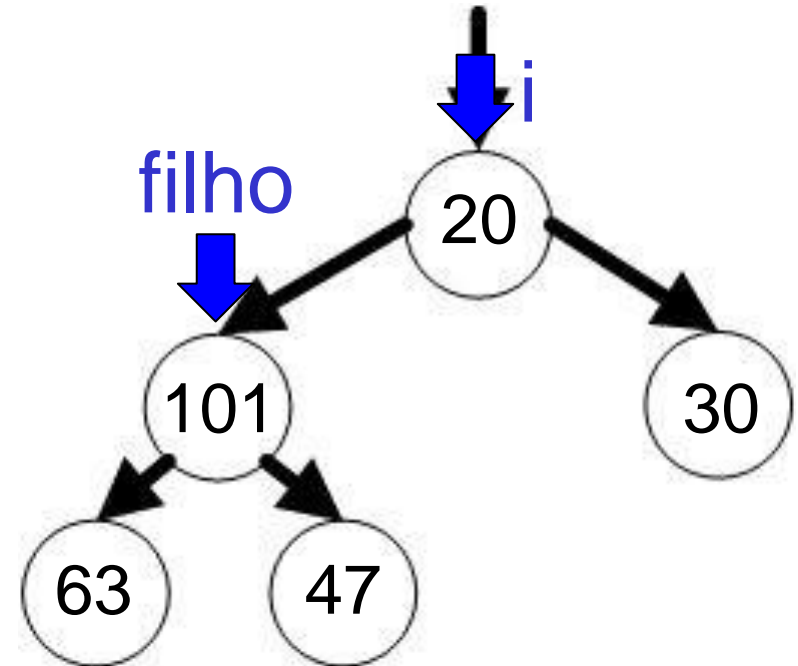
```

tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



↓ i ↓ filho

20	101	30	63	47	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

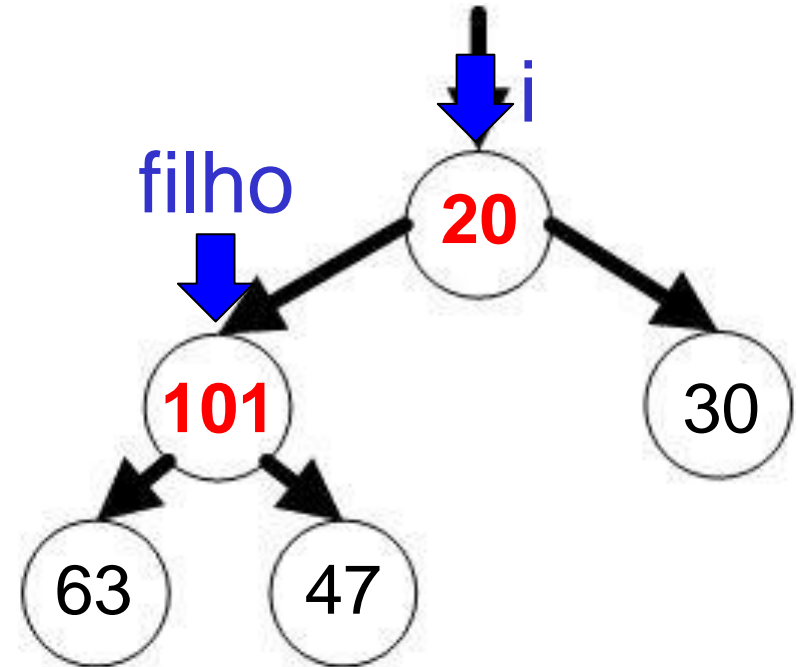
tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 20 < 101



↓ i ↓ filho

20	101	30	63	47	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

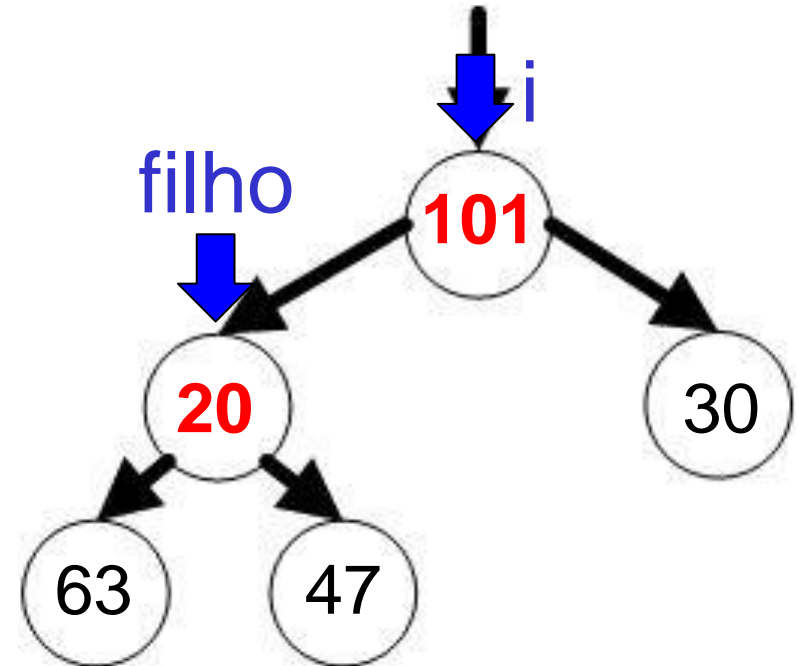
```

tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



↓ i ↓ filho

101	20	30	63	47	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }

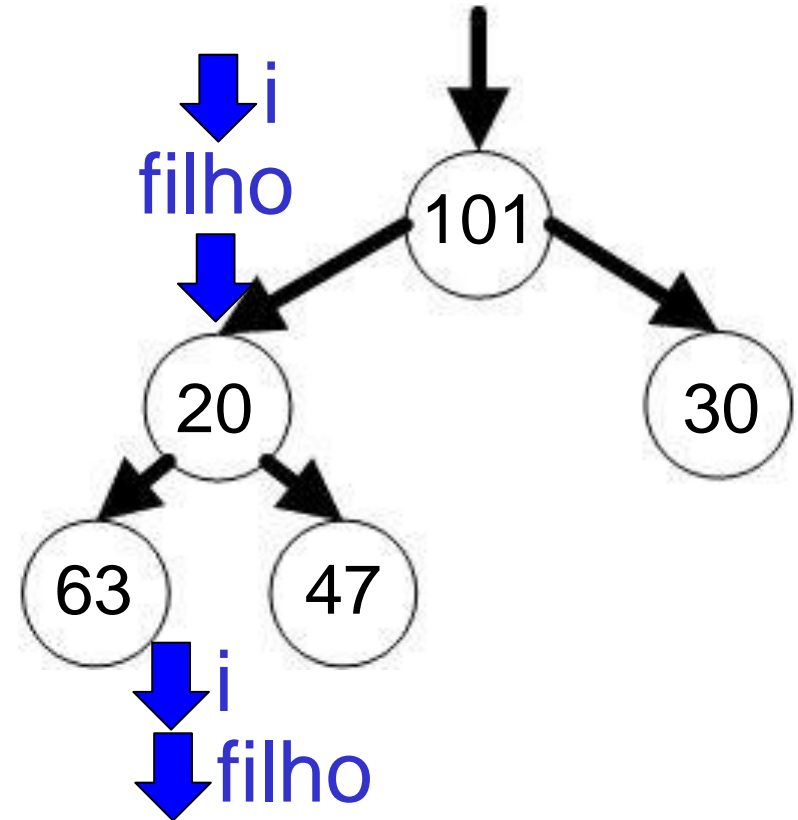
```

tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



101	20	30	63	47	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

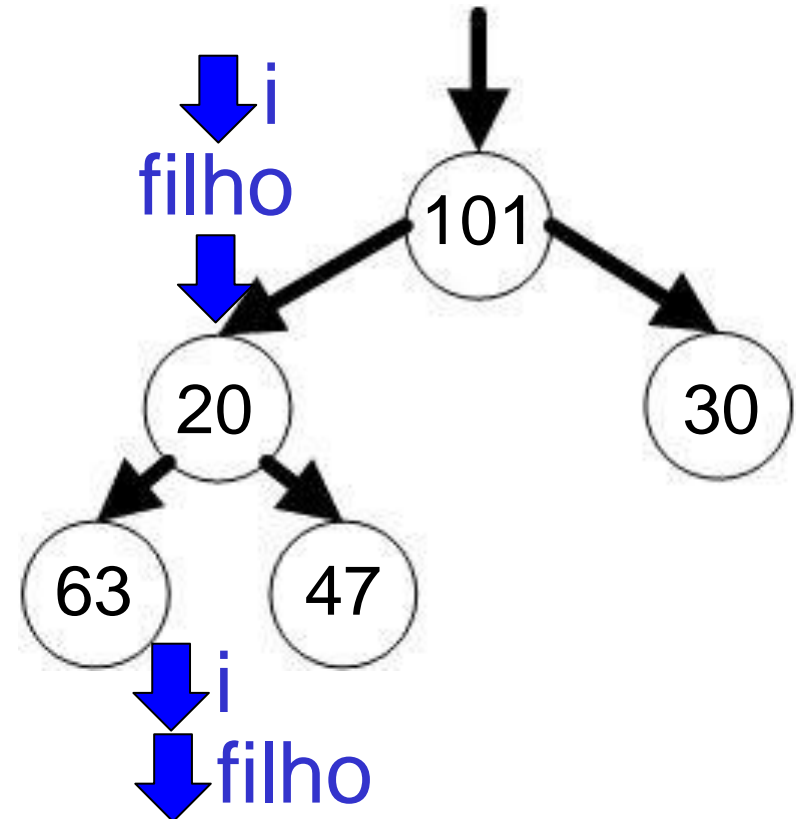
tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 2 <= 2



101	20	30	63	47	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

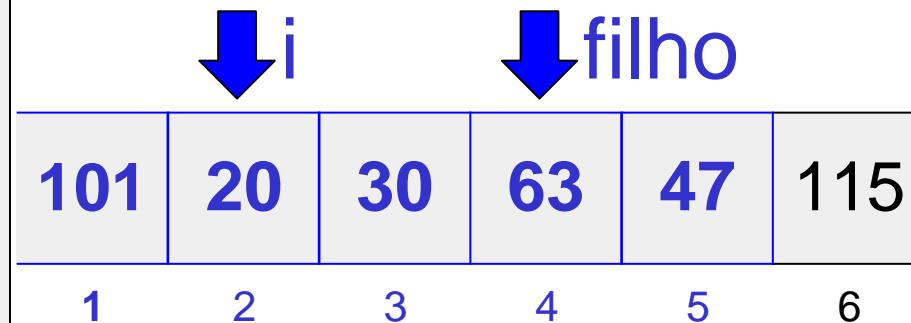
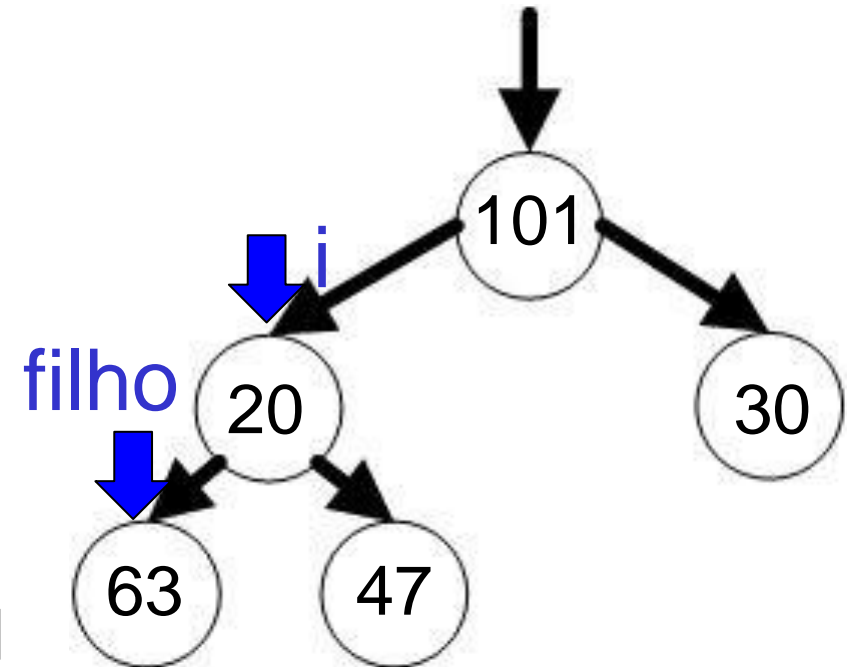
```

tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

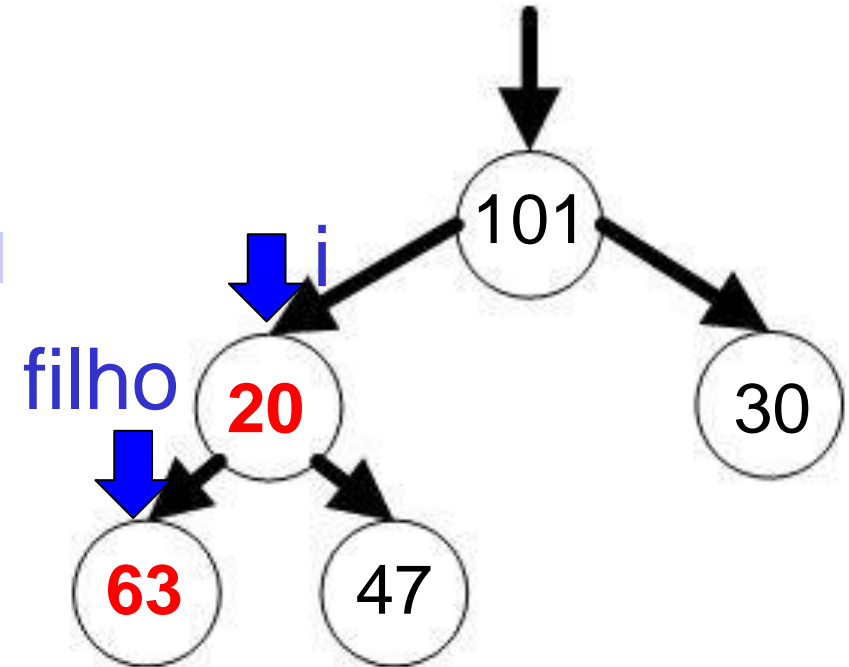
tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 20 < 63



↓ i		↓ filho			
101	20	30	63	47	115
1	2	3	4	5	6

Algoritmo

```

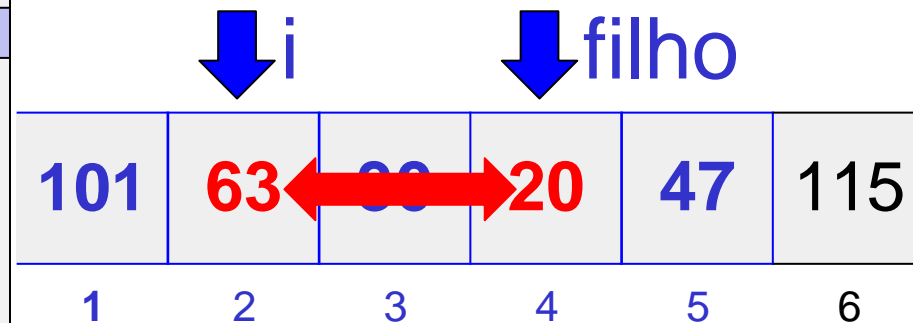
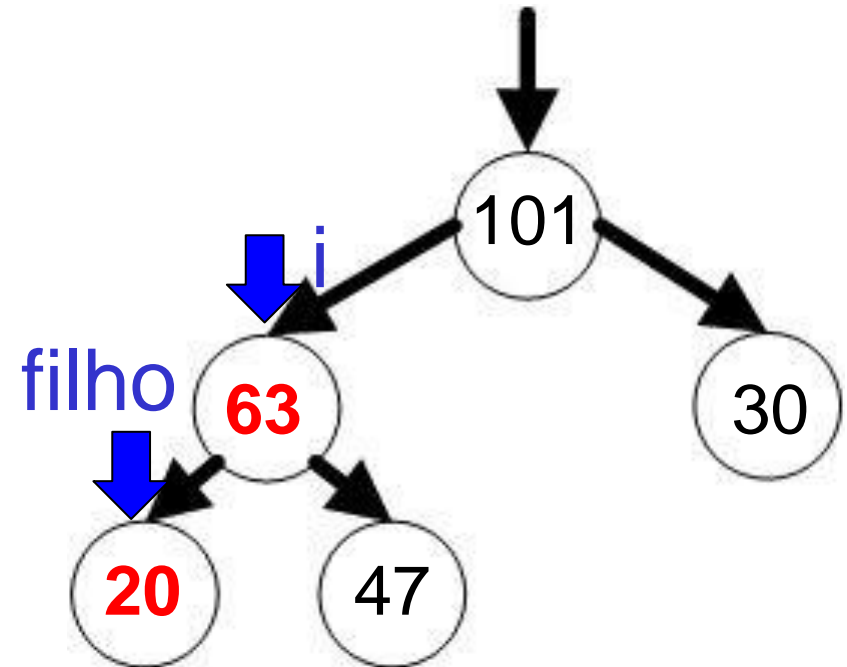
    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }
}
    
```

tam

5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
    
```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }
}

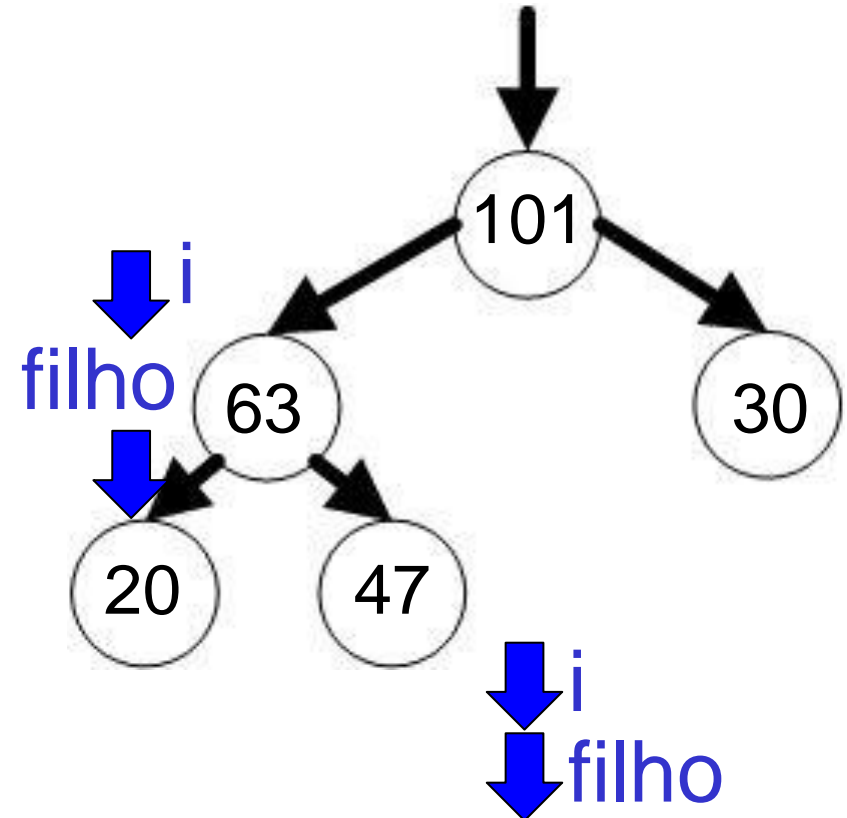
```

tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



101	63	30	20	47	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

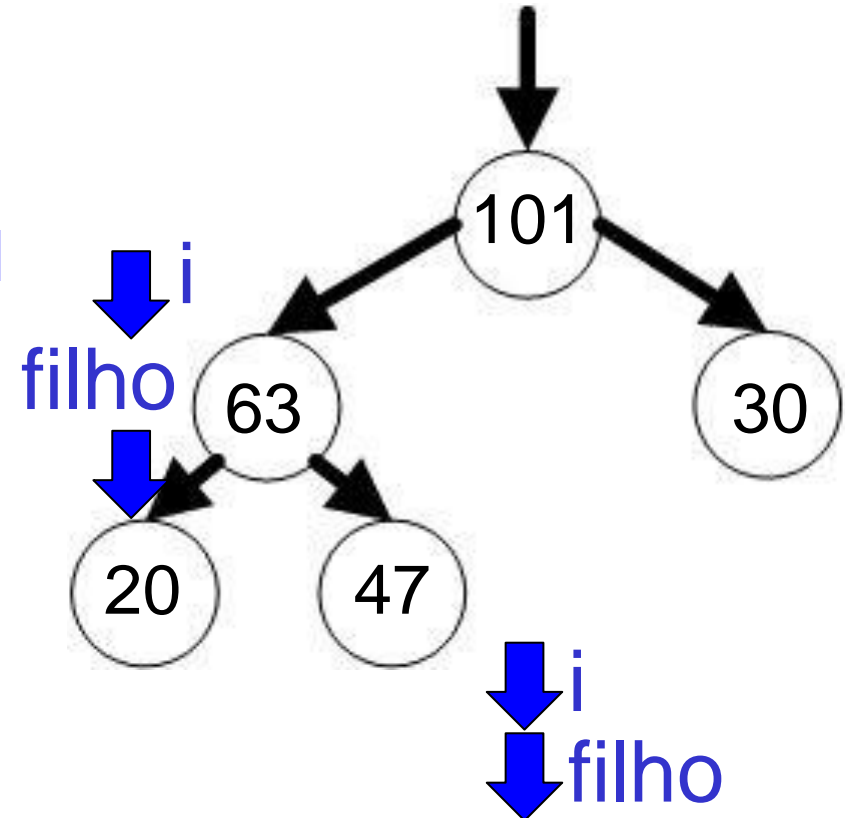
tam 5

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

false: $4 \leq 2$



101	63	30	20	47	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }
}

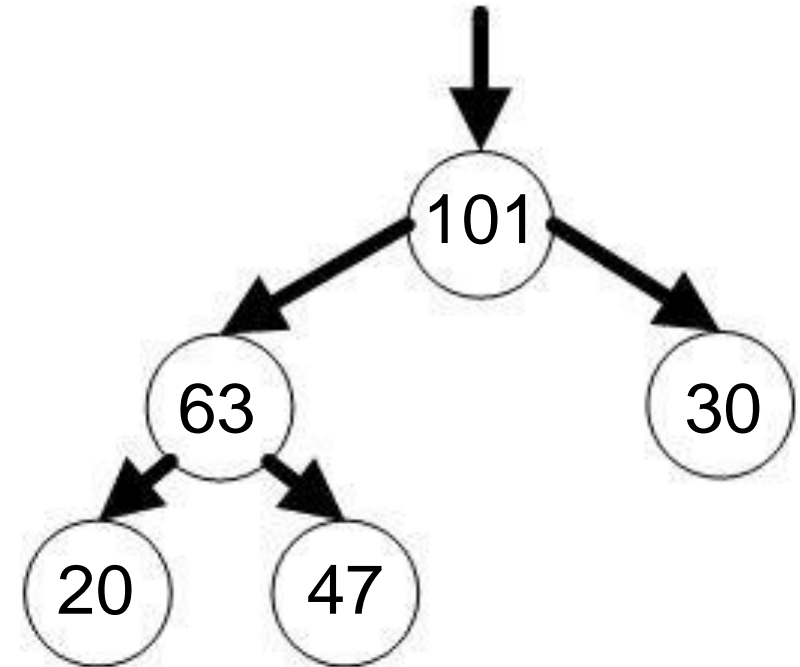
```

tam **5**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



101	63	30	20	47	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }
    }

```

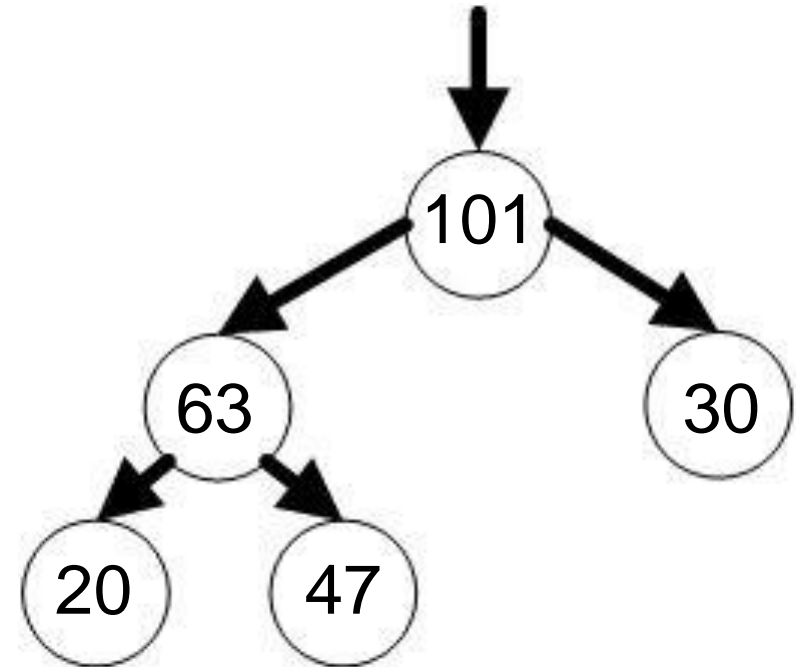
tam: 5

true: 5 > 1

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



101	63	30	20	47	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

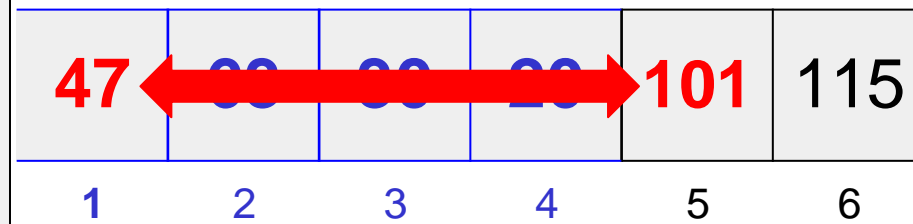
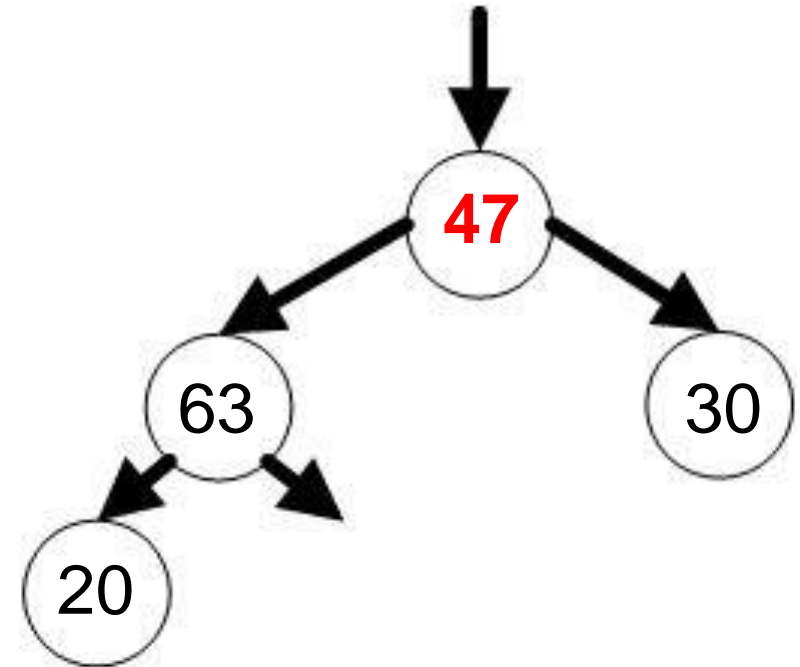
tam

4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

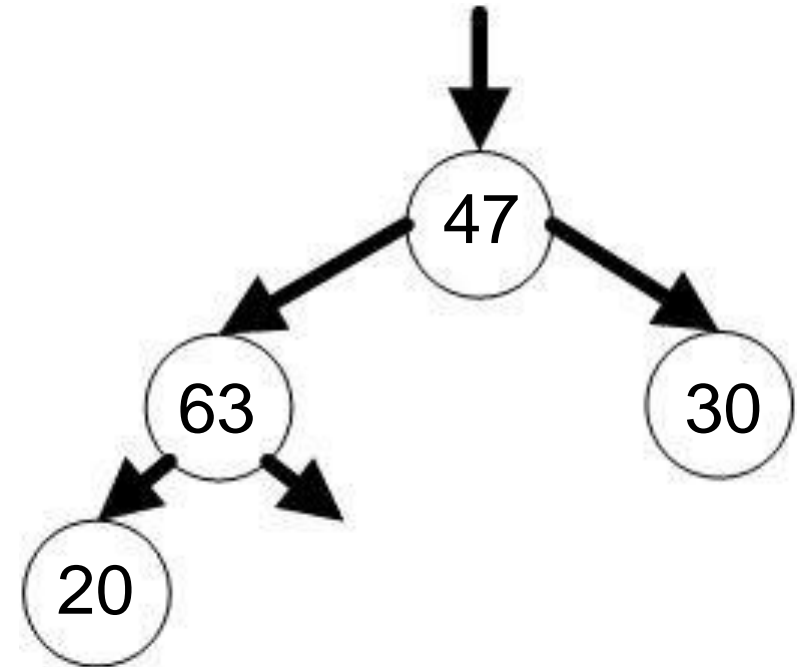
tam

4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



47	63	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }
}

```

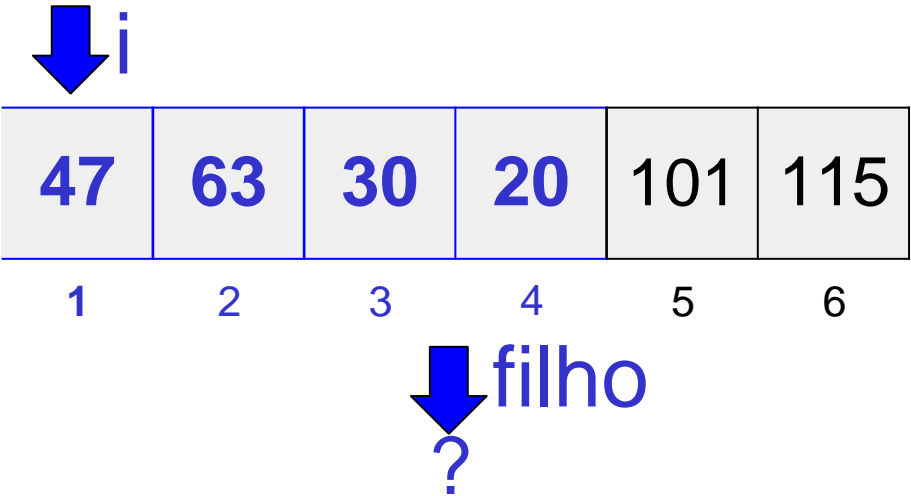
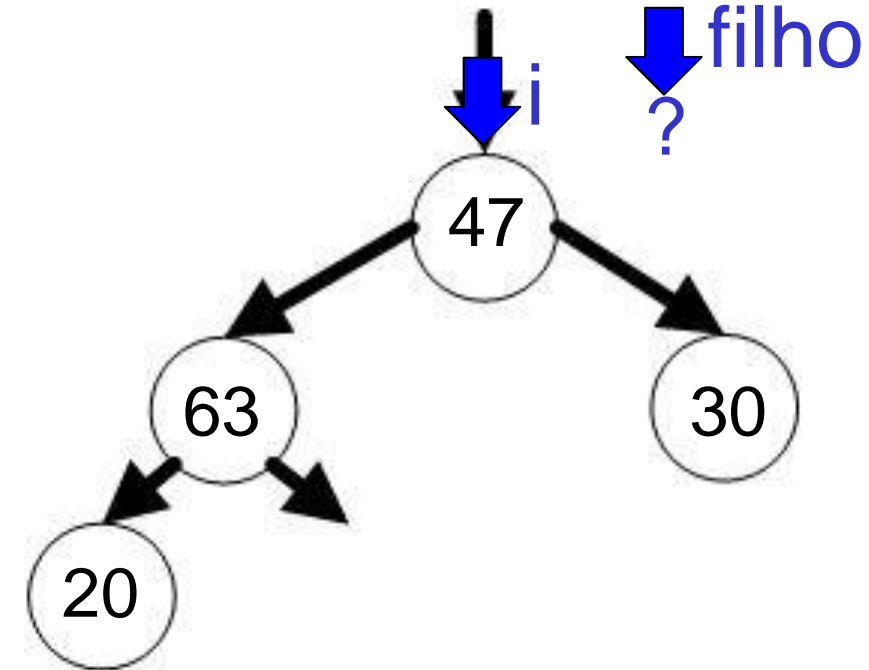
tam

4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }

```

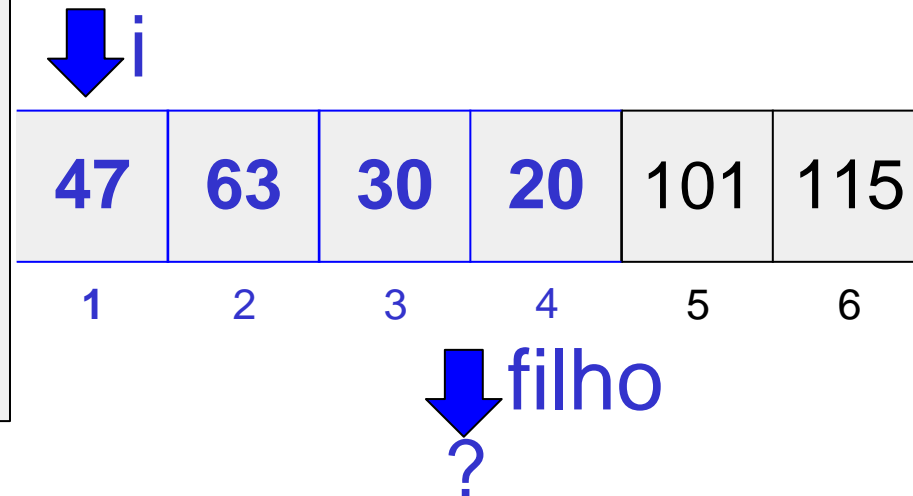
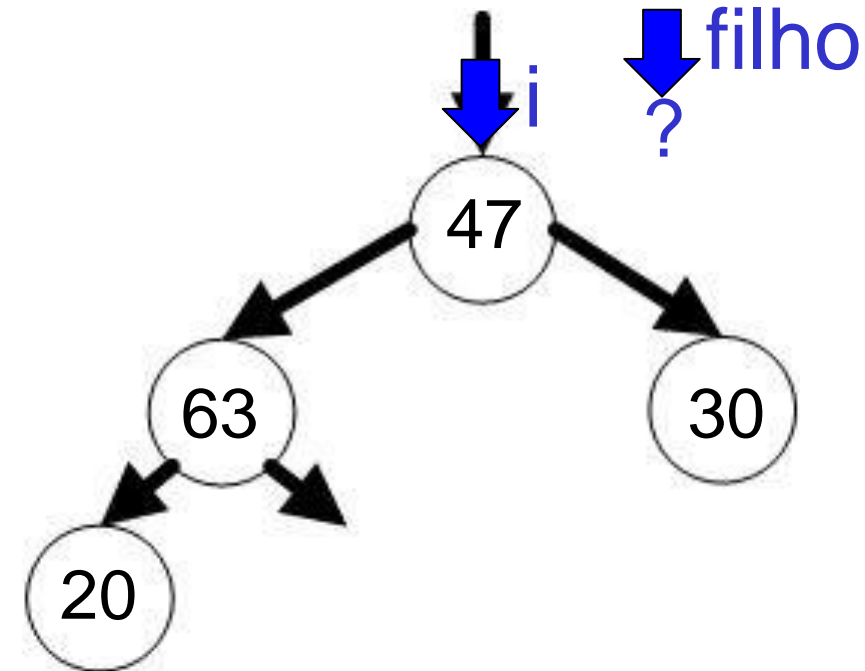
tam 4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 1 <= 2



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

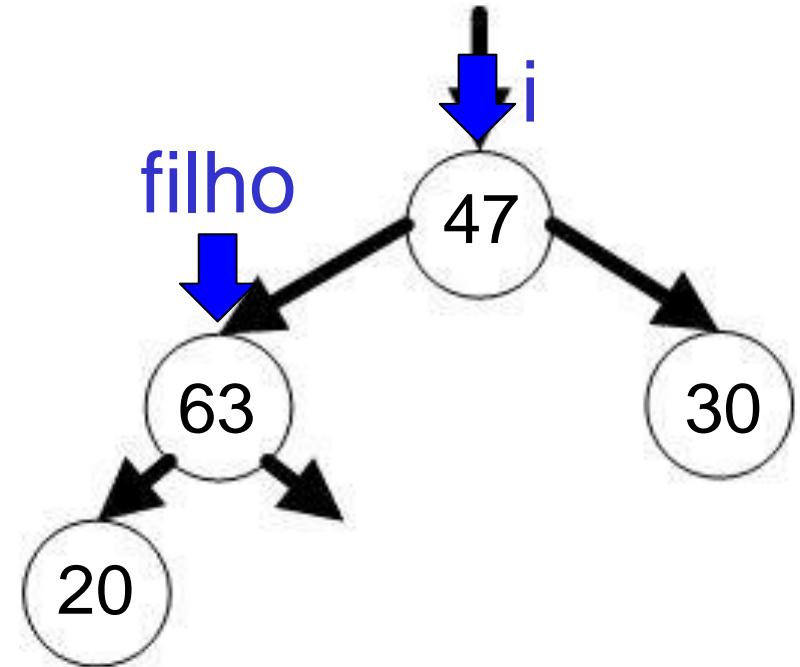
tam

4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



↓ i ↓ filho

47	63	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

tam

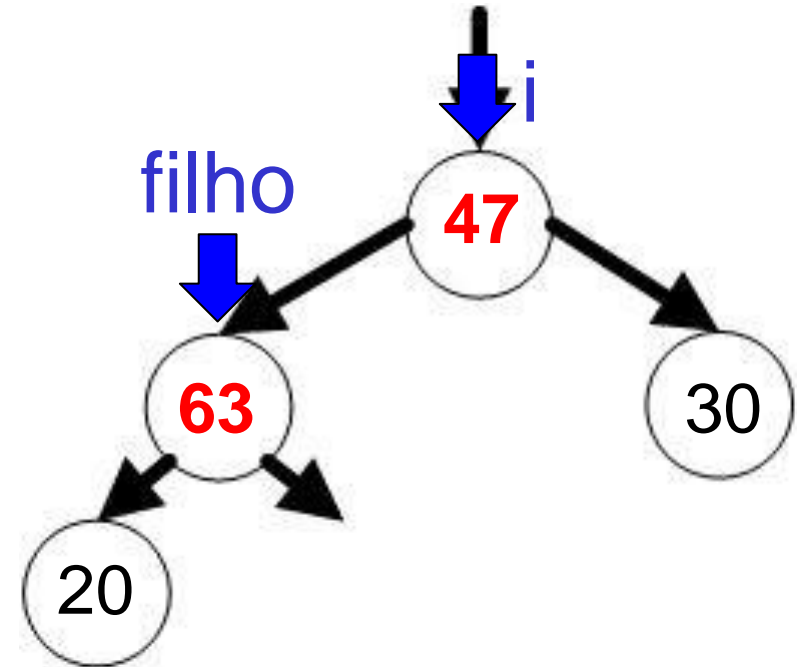
4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 47 < 63



↓ i ↓ filho

47	63	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

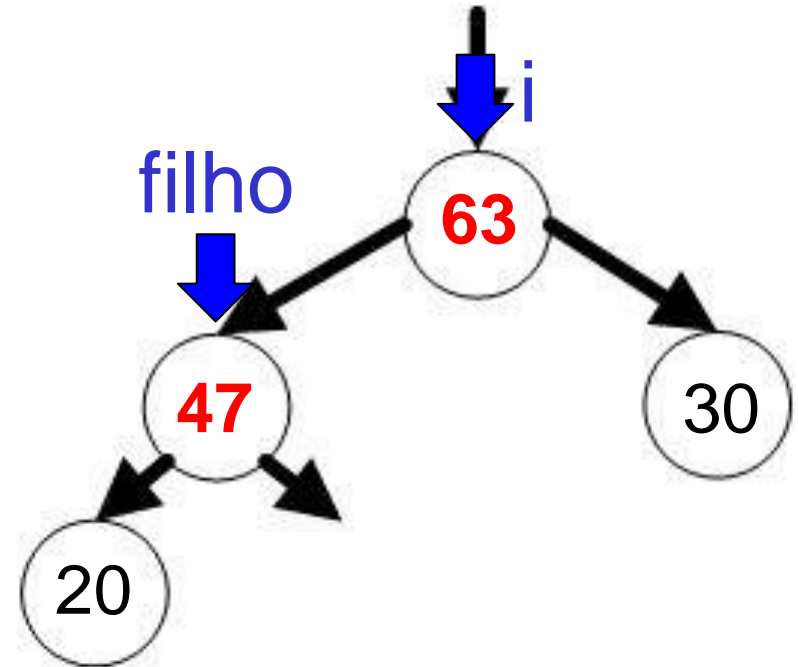
tam

4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



↓ i ↓ filho

63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }

```

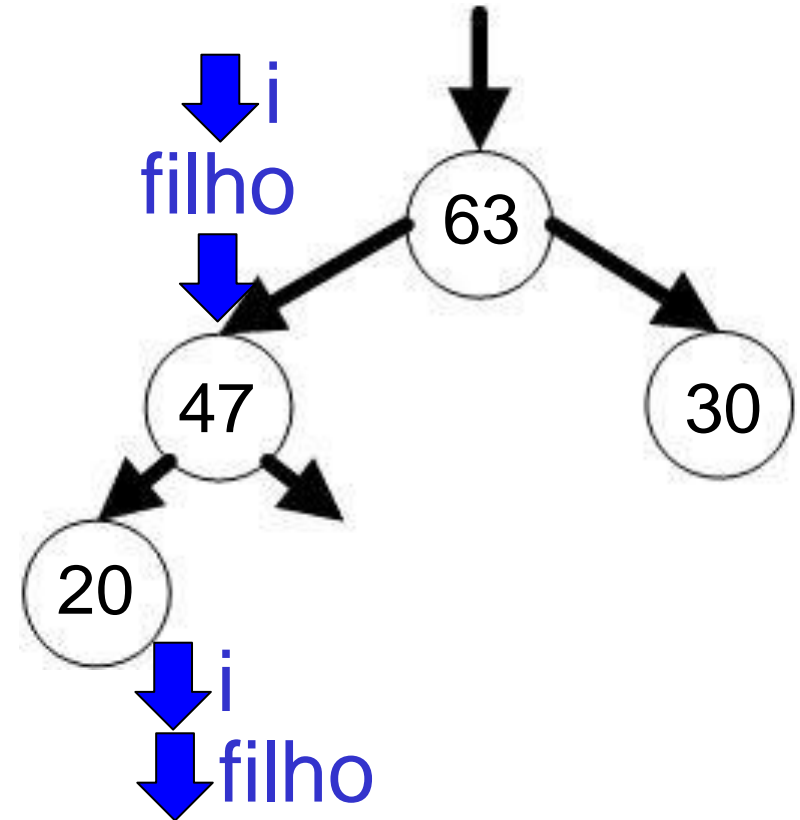
tam

4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

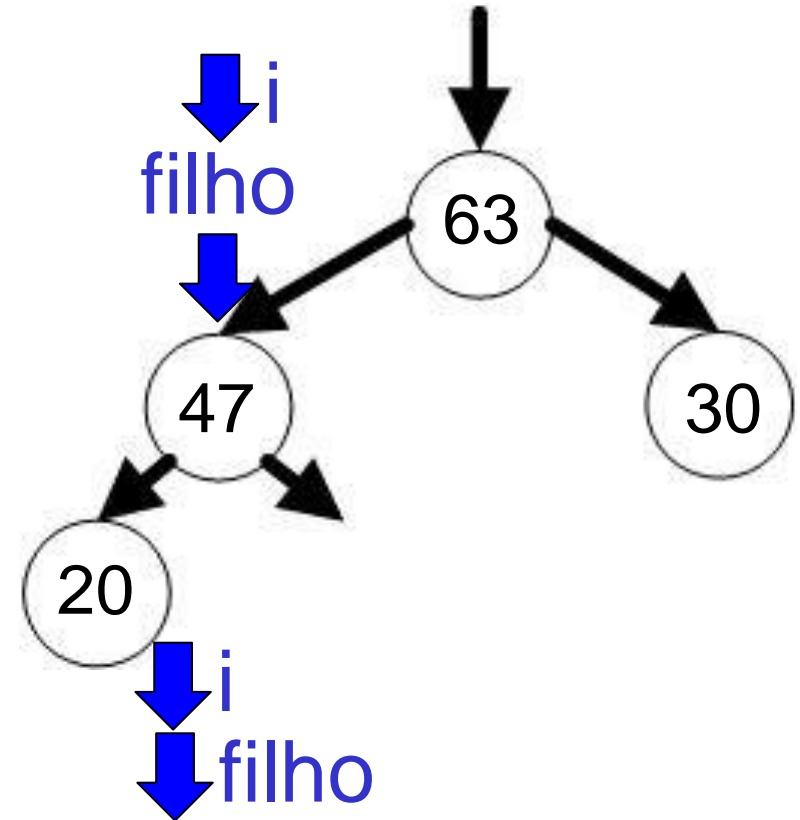
tam 4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: $2 \leq 2$



63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

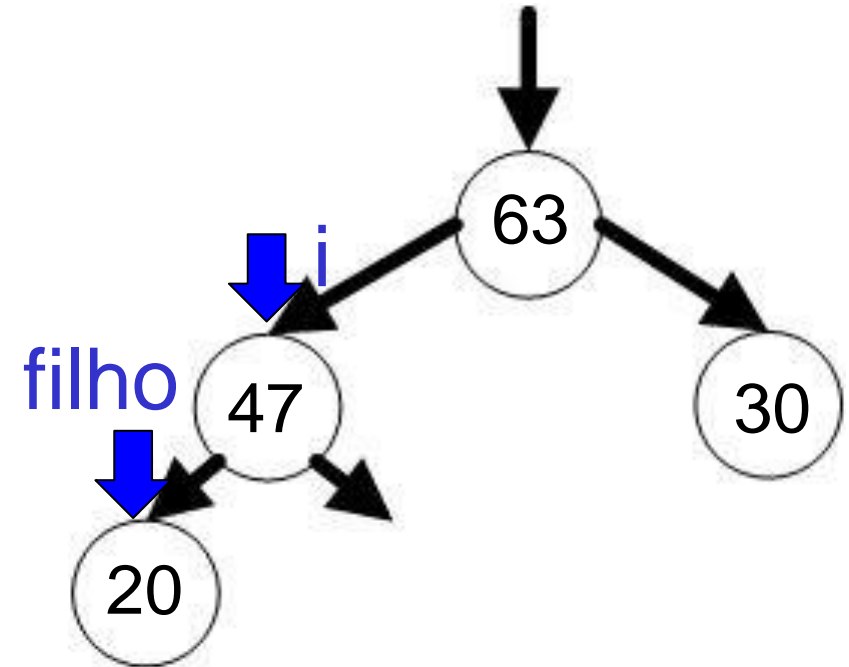
tam

4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



↓ i		↓ filho			
63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

tam

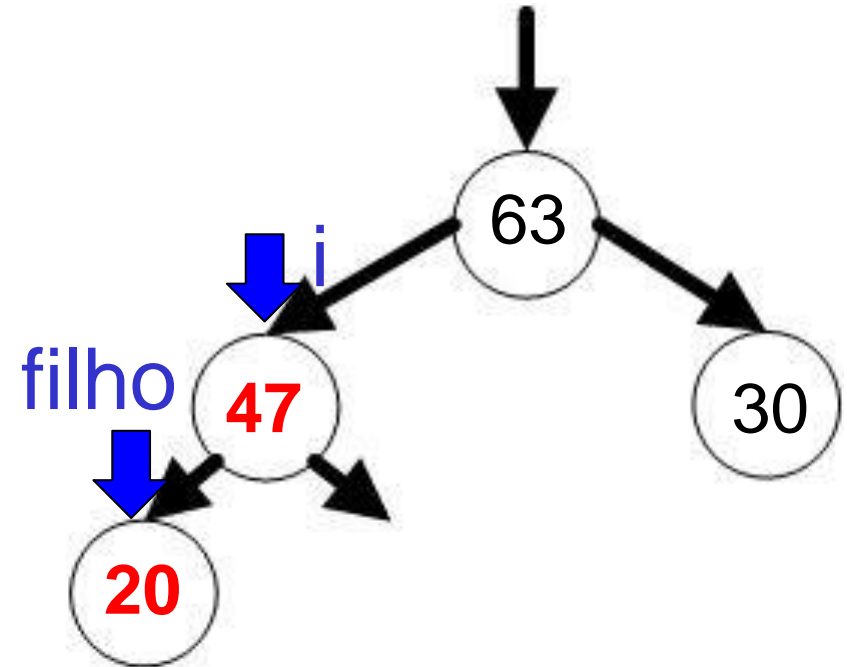
4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

false: 47 < 20



↓ i		↓ filho			
63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

tam

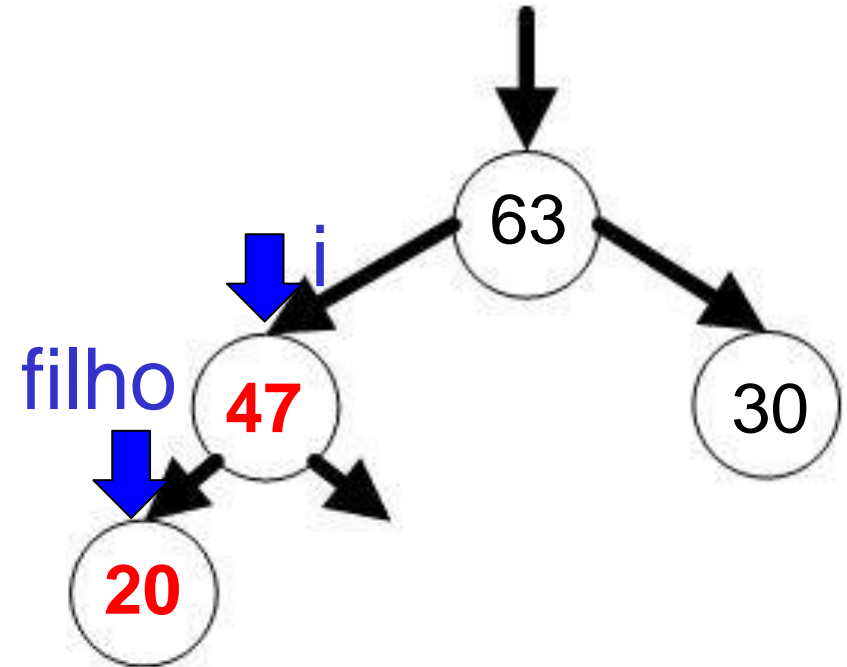
4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

false: 47 < 20



↓ i		↓ filho			
63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

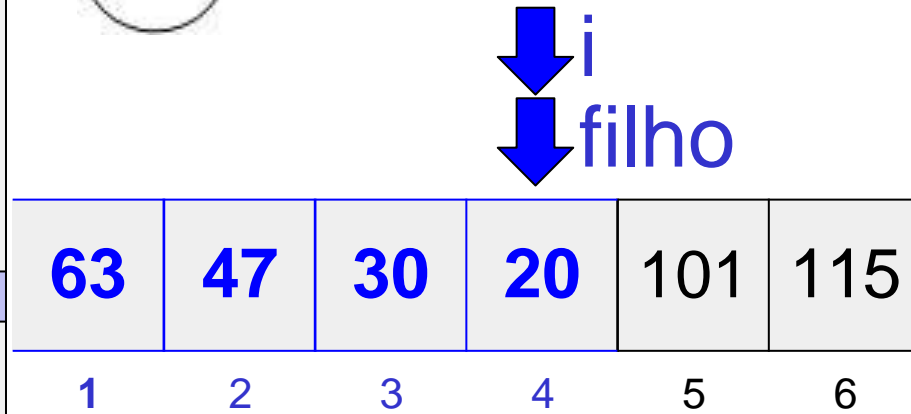
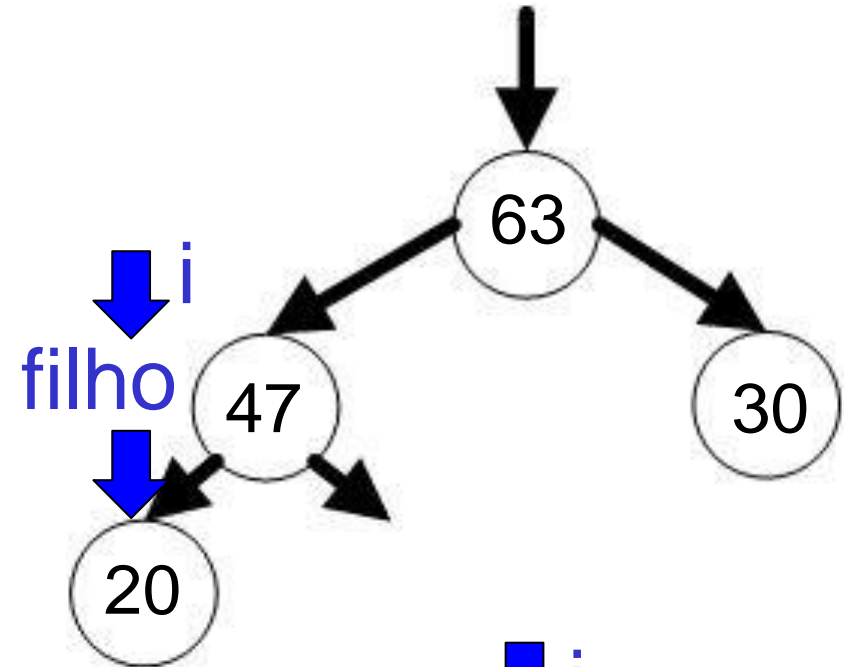
tam

4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

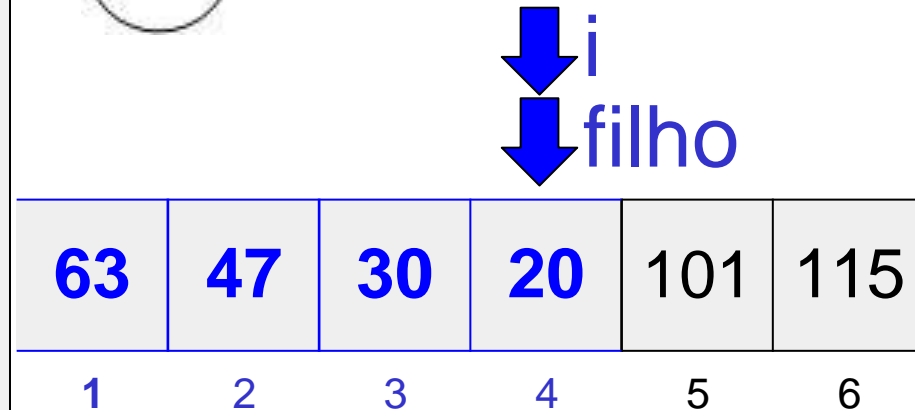
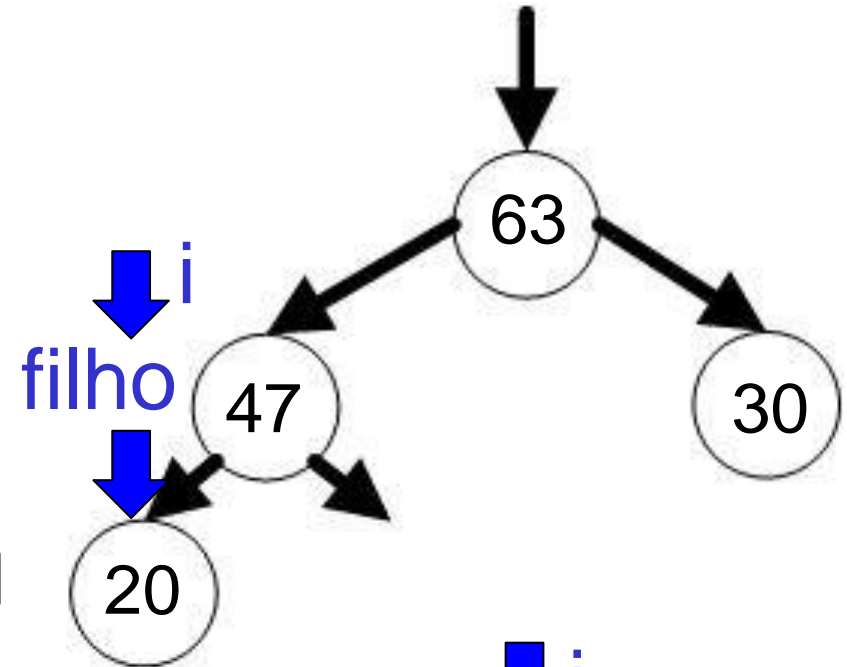
tam 4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

false: $4 \leq 2$



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }

```

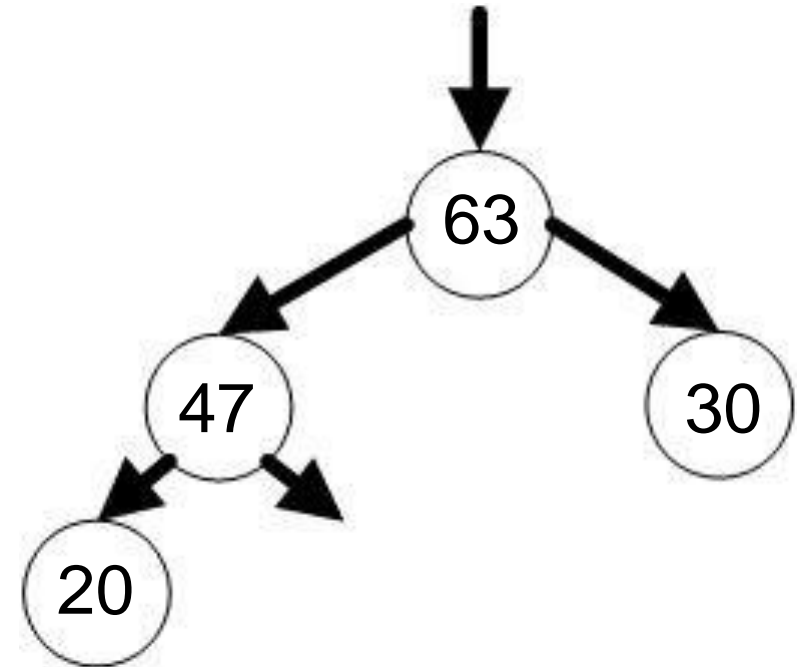
tam

4

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }
    }

```

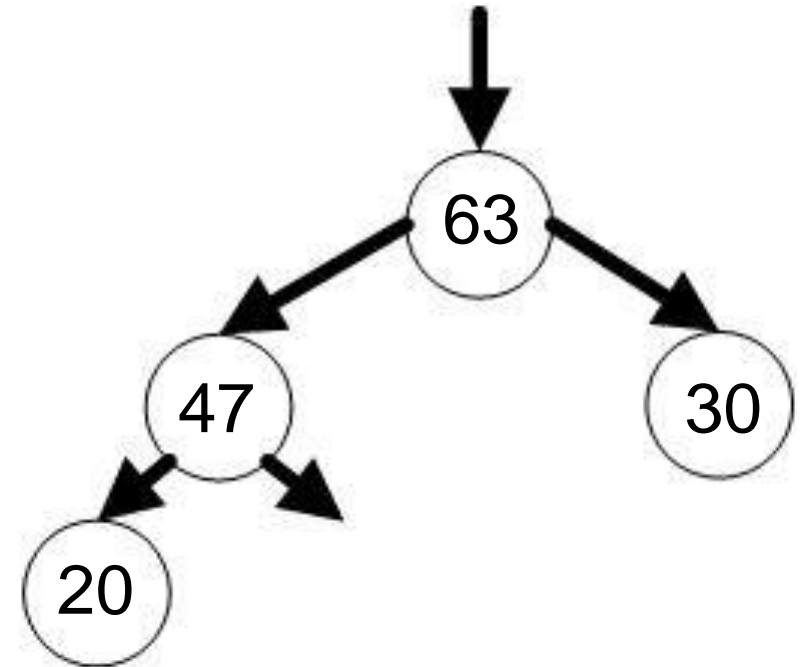
tam: 4

true: 4 > 1

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



63	47	30	20	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

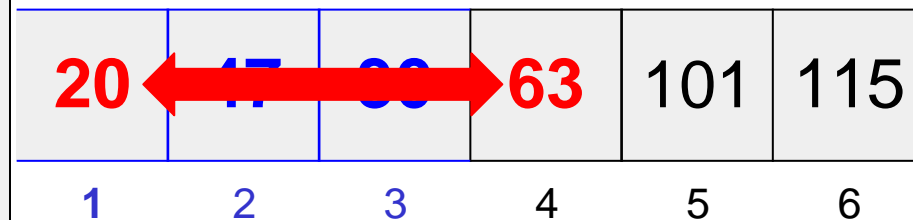
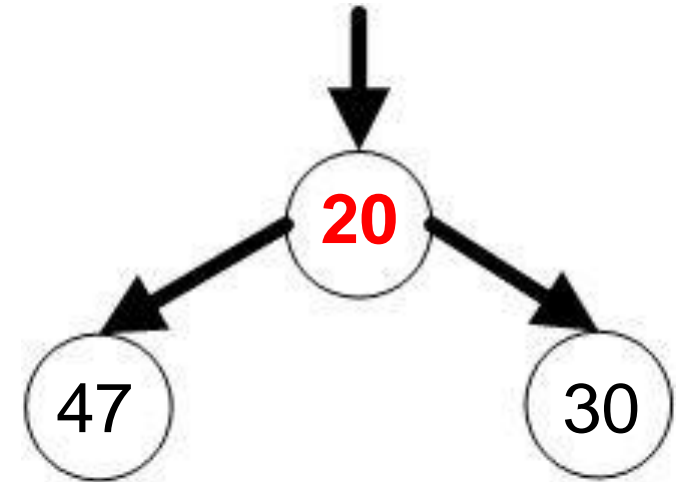
```

tam **3**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }
}

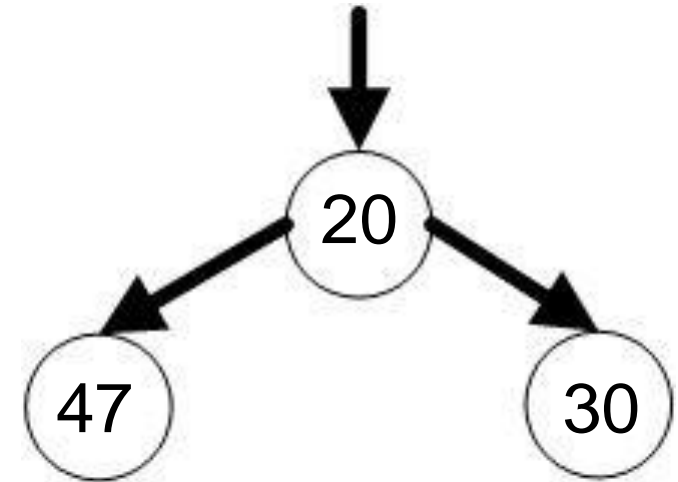
```

tam **3**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



20	47	30	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

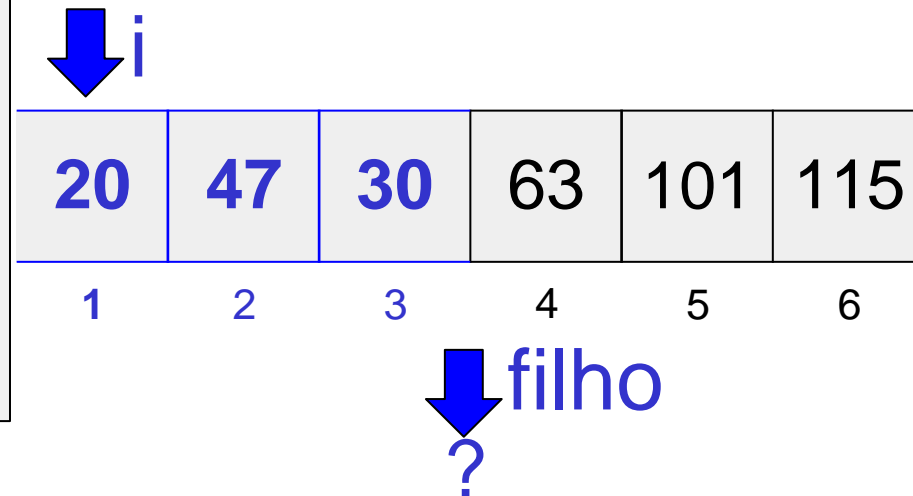
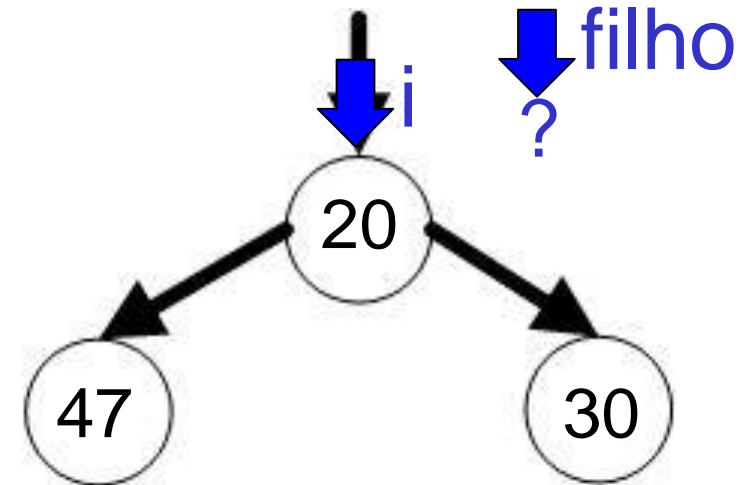
```

tam **3**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }

```

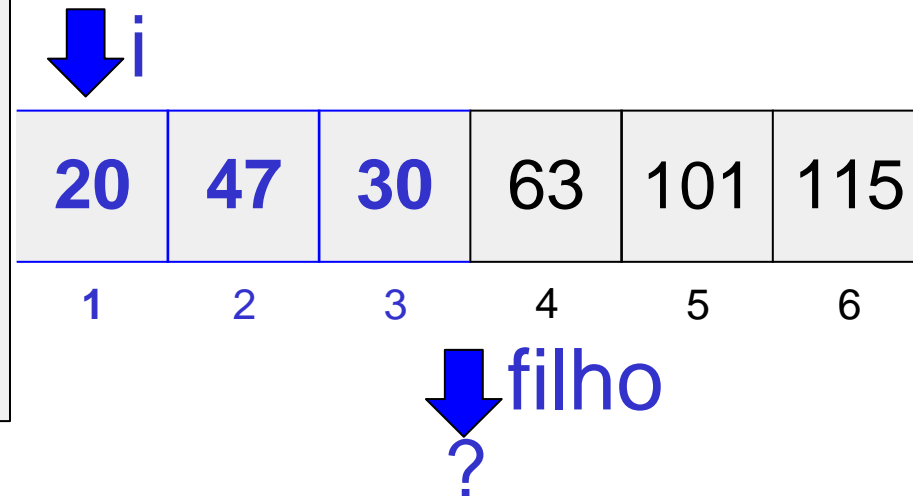
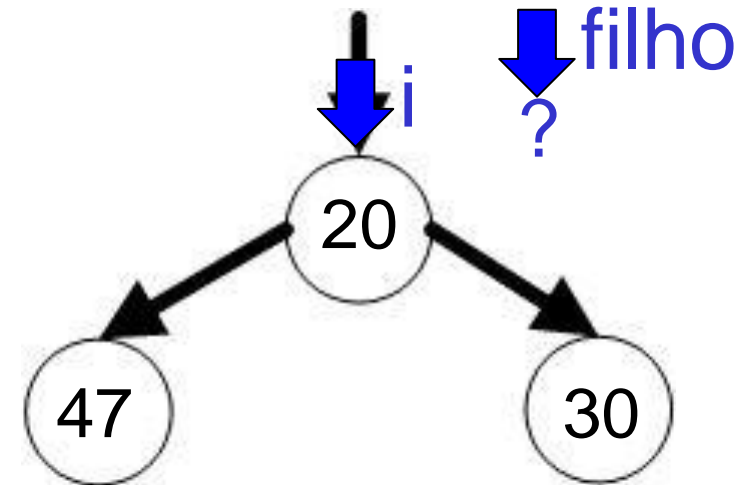
tam 3

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 1 <= 1



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

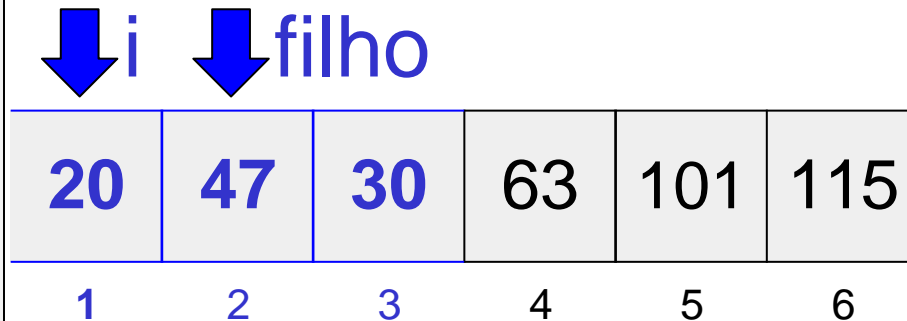
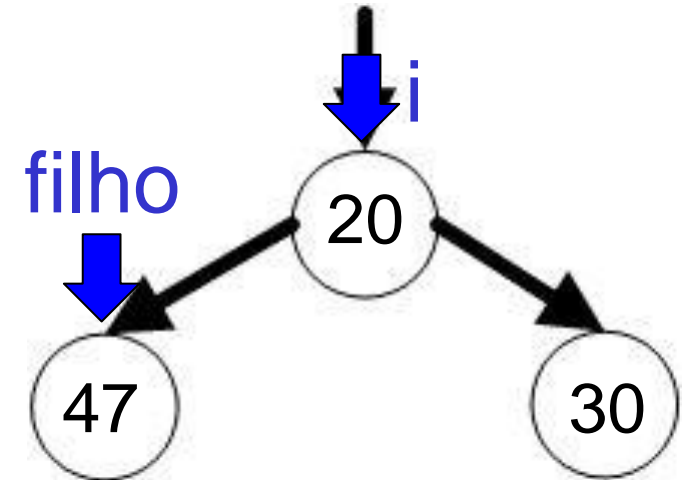
```

tam **3**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

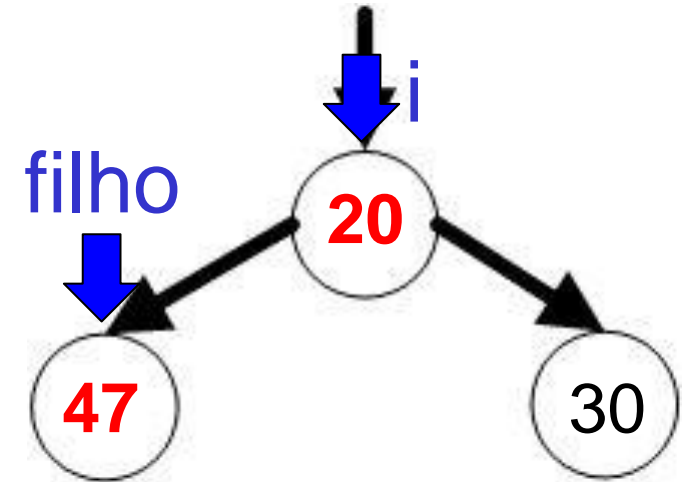
tam **3**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: 20 < 47



↓ i		↓ filho			
20	47	30	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

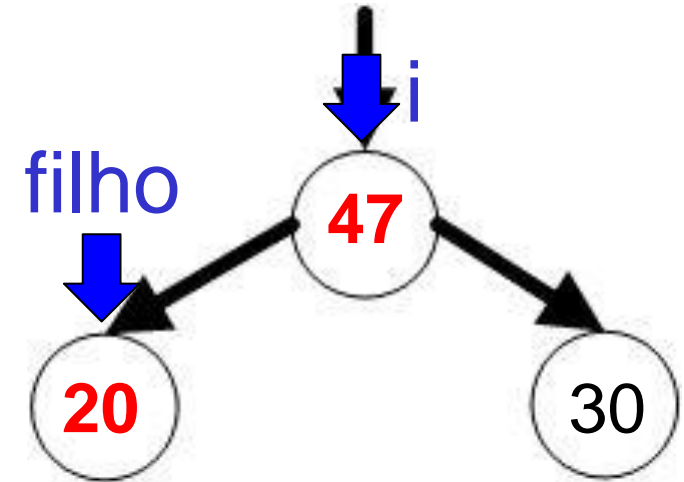
```

tam **3**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



i **filho**

47	20	30	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

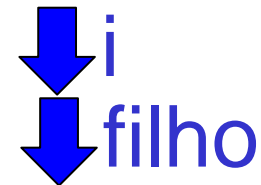
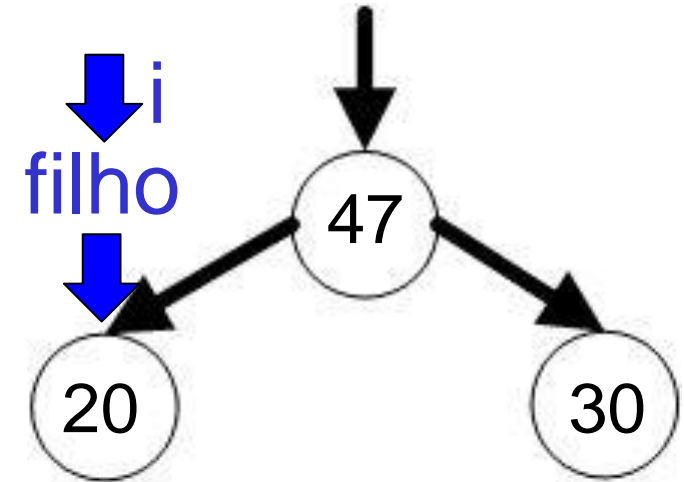
```

tam **3**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



47	20	30	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }

```

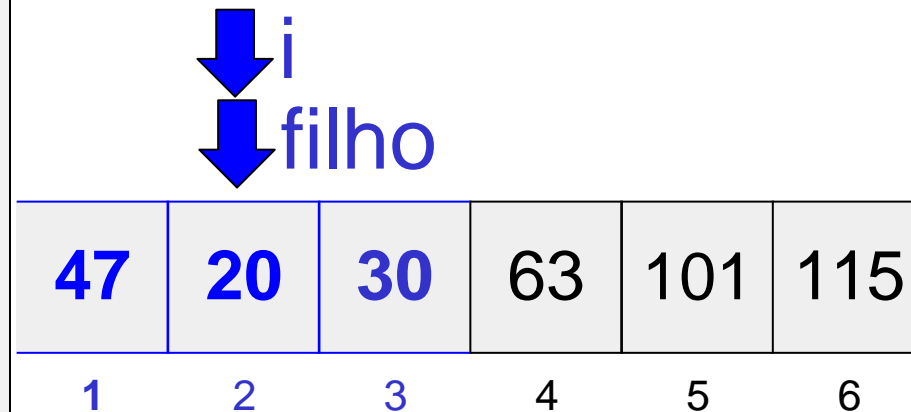
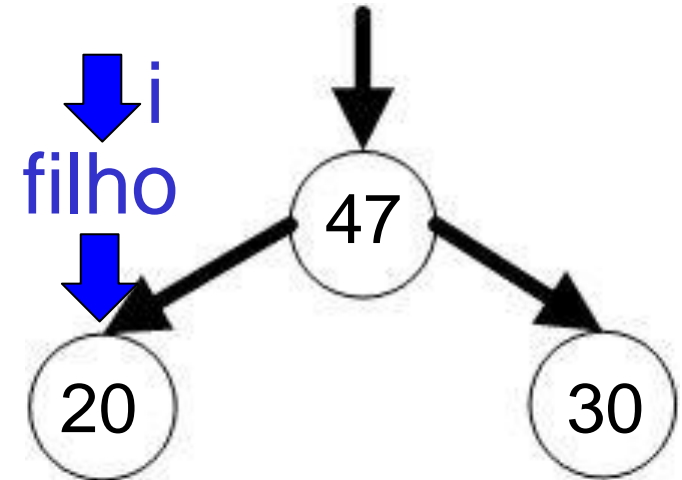
tam 3

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

false: $2 \leq 1$



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }
}

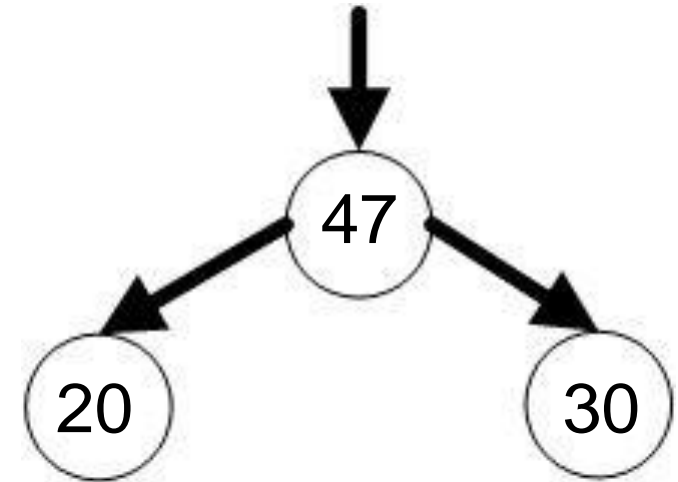
```

tam **3**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



47	20	30	63	101	115
1	2	3	4	5	6

Algoritmo

```

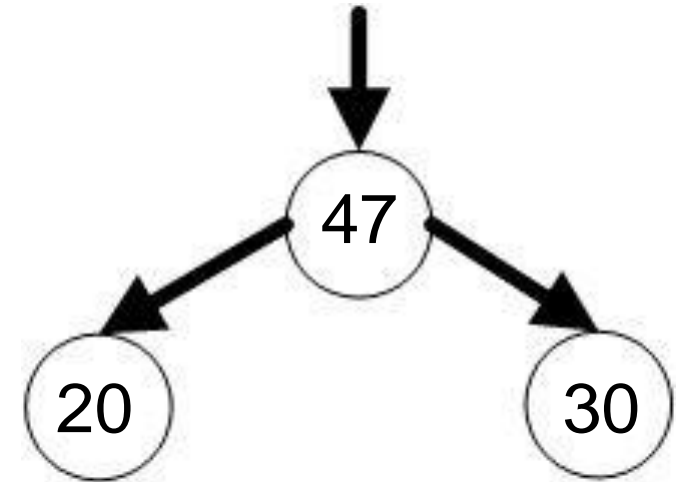
    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }
    }
    true: 3 > 1

```

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



47	20	30	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

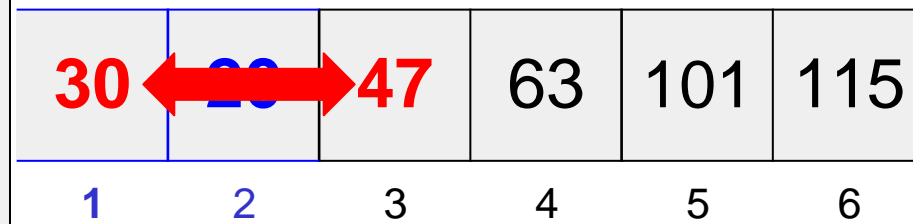
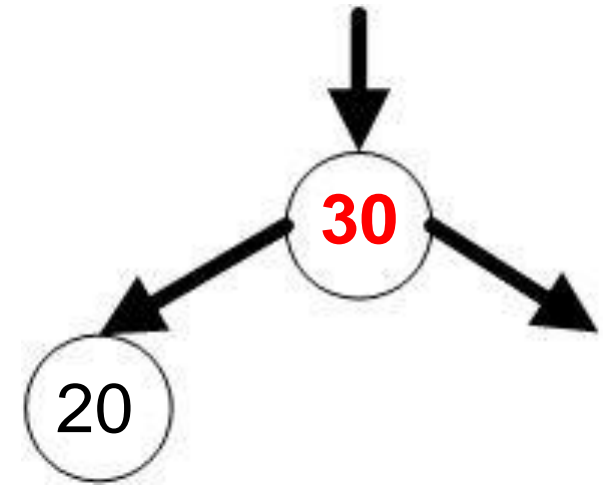
```

tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

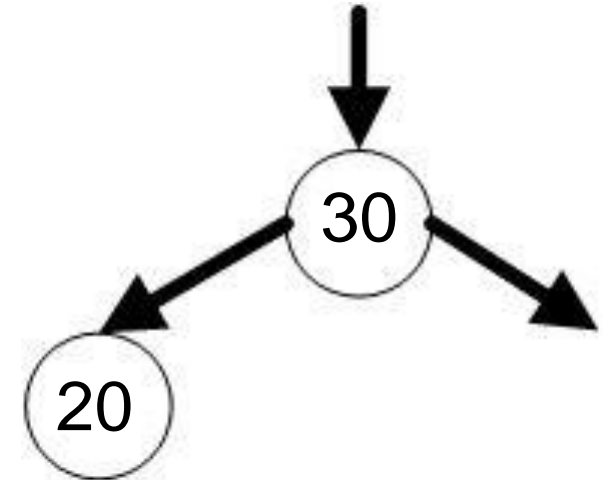
```

tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }

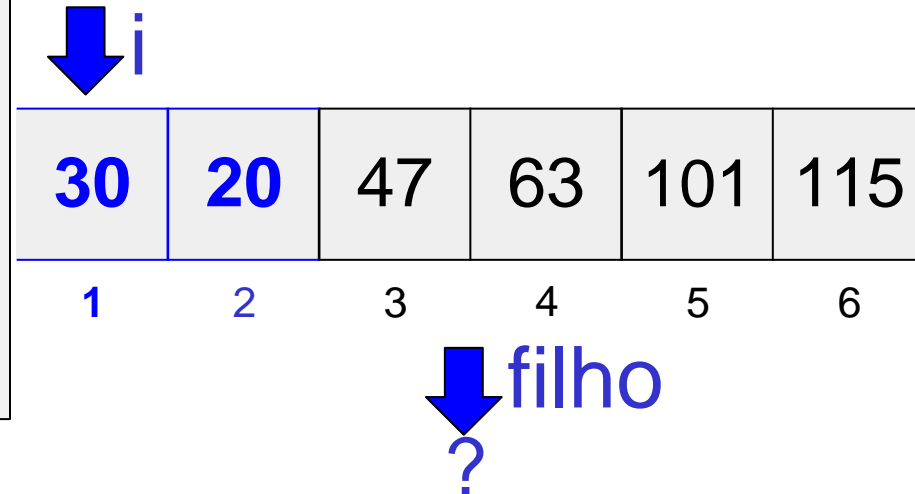
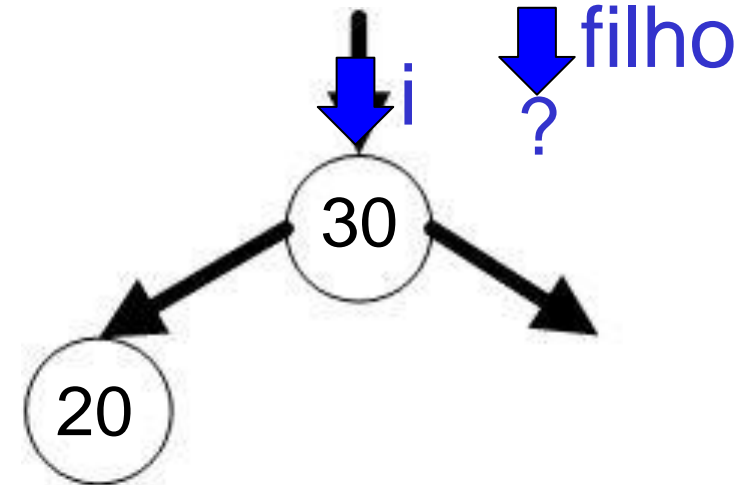
```

tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

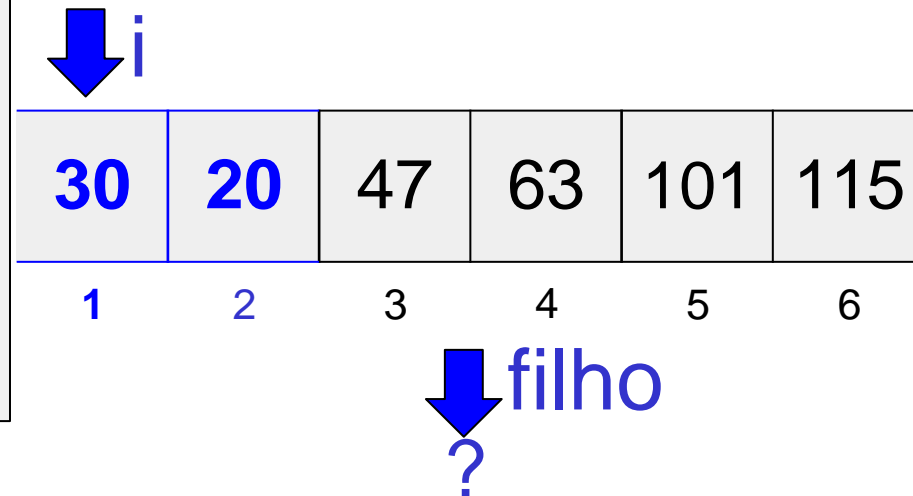
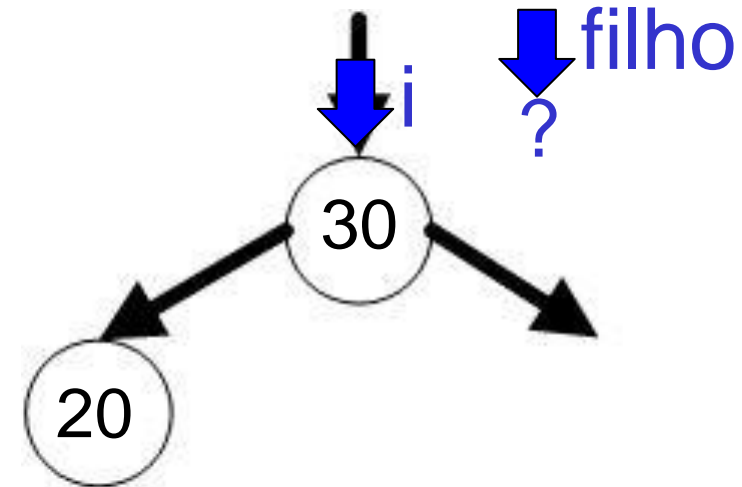
tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

true: $1 \leq 1$



Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

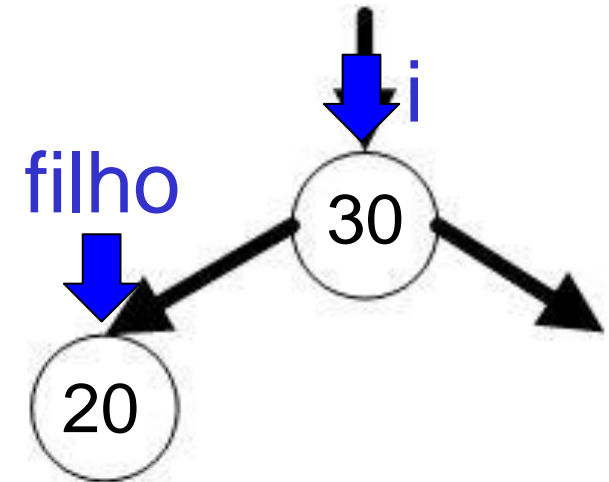
```

tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

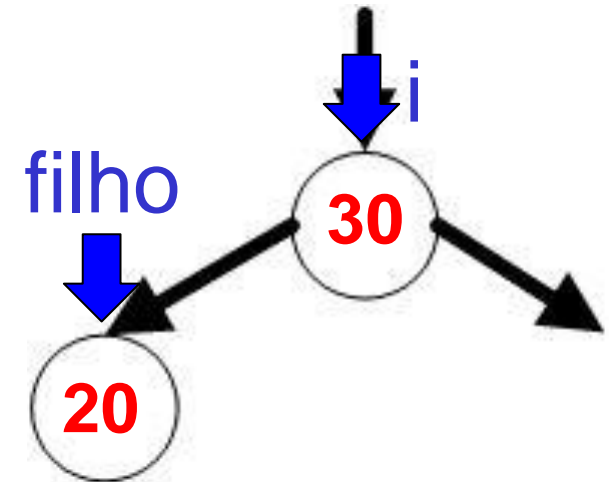
tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

false: $30 < 20$



30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

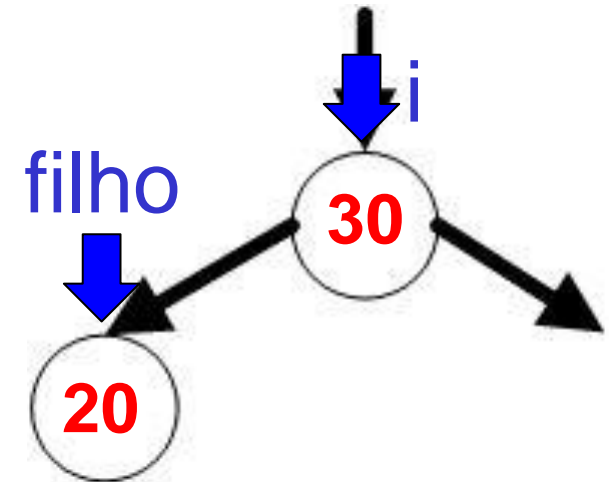
tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

false: $30 < 20$



i **filho**

30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

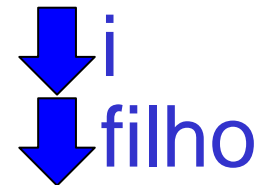
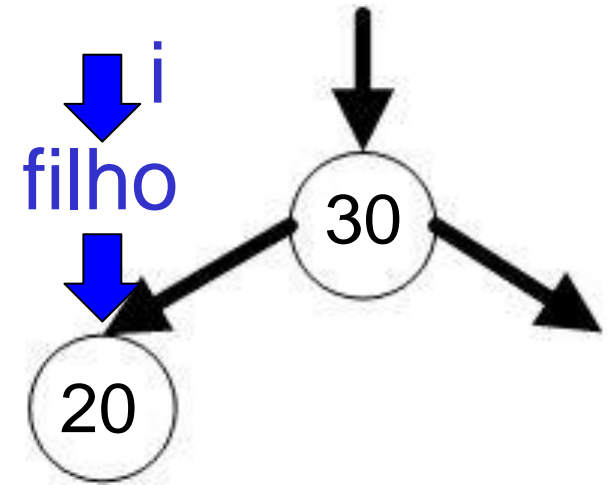
```

tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

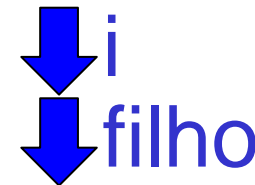
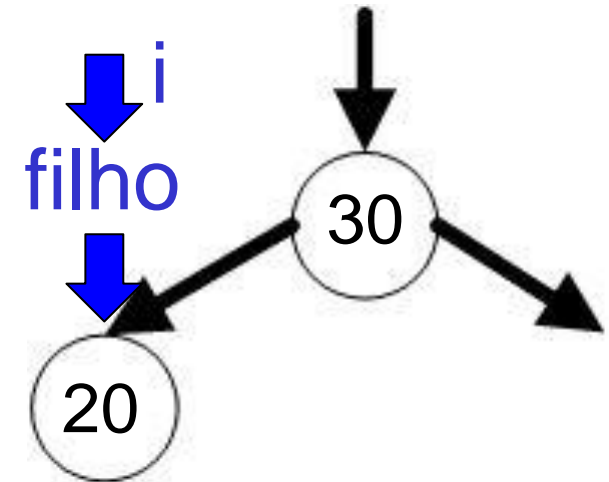
tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```

false: $2 \leq 1$



30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

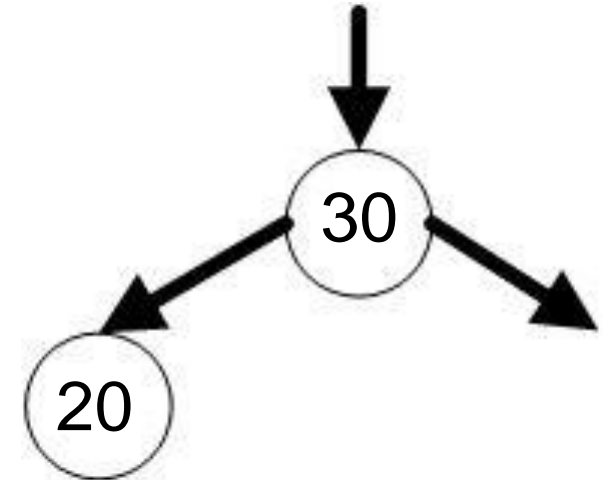
```

tam **2**

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



30	20	47	63	101	115
1	2	3	4	5	6

Algoritmo

```
tam = n;
```

tam

2

```
while (tam > 1){
```

```
    swap(1, tam--);
```

```
    Reconstruir(array, tam);
```

```
}
```

```
}
```

true: $2 > 1$

```
void Reconstruir(int[] array, int tam){
```

```
    int i = 1;
```

```
    while (HasFilho(i, tam) == true){
```

```
        int filho = GetMaiorFilho(i, tam);
```

```
        if (array[i] < array[filho]) {
```

```
            swap(i, filho);
```

```
            i = filho;
```

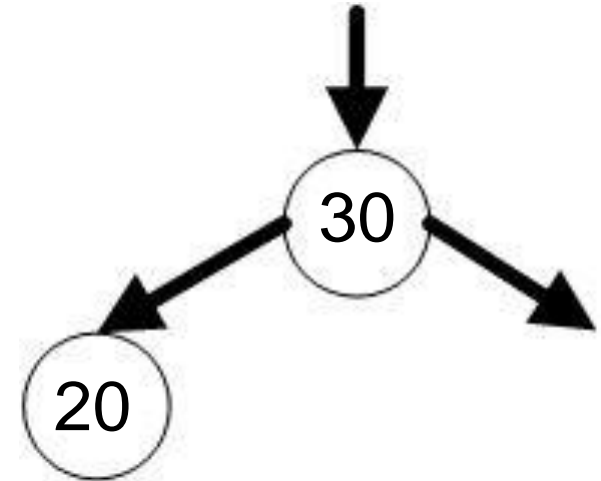
```
        } else {
```

```
            i = tam;
```

```
        }
```

```
    }
```

```
}
```



30

20

47

63

101

115

1

2

3

4

5

6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



20	30	47	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

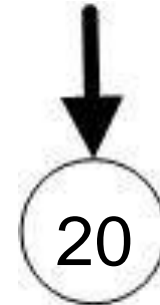
tam

1

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



20	30	47	63	101	115
1	2	3	4	5	6

Algoritmo

```

    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array, tam);
    }

```

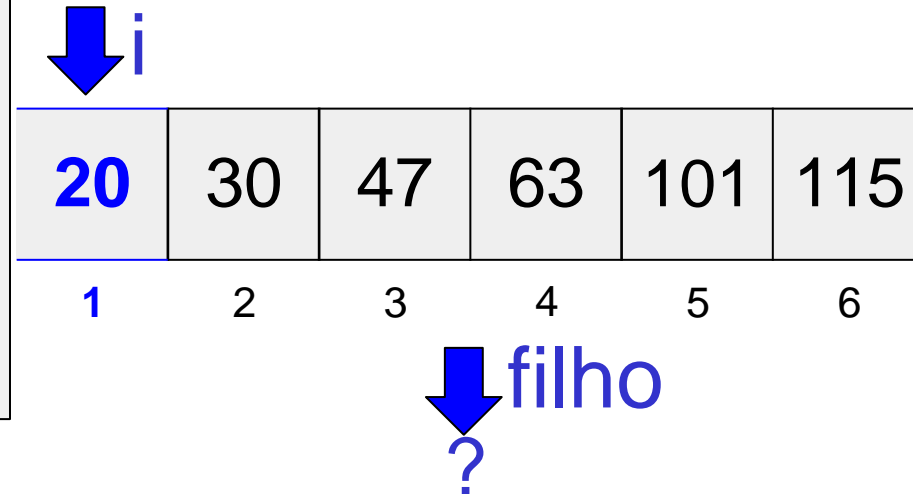
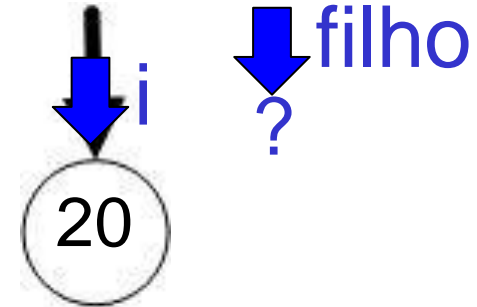
tam

1

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



Algoritmo

...

```
tam = n;
while (tam > 1){
    swap(1, tam--);
```

```
    Reconstruir(array, tam);
```

tam

1

```
void Reconstruir(int[] array, int tam){
```

```
    int i = 1;
```

false: $1 \leq 0$

```
    while (HasFilho(i, tam) == true){
```

```
        int filho = GetMaiorFilho(i, tam);
```

```
        if (array[i] < array[filho]) {
```

```
            swap(i, filho);
```

```
            i = filho;
```

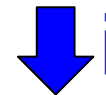
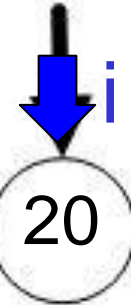
```
        } else {
```

```
            i = tam;
```

```
        }
```

```
    }
```

```
}
```



20	30	47	63	101	115
1	2	3	4	5	6



Algoritmo

```

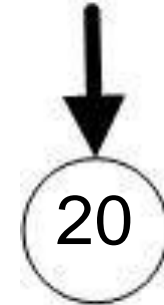
    ...
    tam = n;
    while (tam > 1){
        swap(1, tam--);
        Reconstruir(array,tam);
    }

```

```

void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}

```



20	30	47	63	101	115
1	2	3	4	5	6

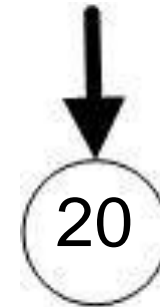
Algoritmo

```

    ...
tam = n;
while (tam > 1){
    swap(1, tam--);
    Reconstruir(array,tam);
}

```

false: 1 > 1



20	30	47	63	101	115
1	2	3	4	5	6

Algoritmo

■ ■ ■

```
tam = n;  
while (tam > 1){  
    swap(1, tam--);  
    Reconstruir(array,tam);  
}
```

20	30	47	63	101	115
1	2	3	4	5	6

Exercício Resolvido

- Implemente os métodos:

int GetMaiorFilho(**int** i, **int** tam) e

boolean HasFilho(**int** i, **int** tam) apresentados anteriormente

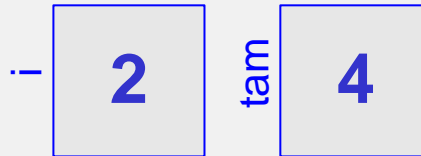
Exercício Resolvido

- Implemente os métodos:

int GetMaiorFilho(**int** i, **int** tam) e

boolean HasFilho(**int** i, **int** tam) apresentados anteriormente

```
int GetMaiorFilho(int i, int tam){  
    int filho;  
    if ( $2*i == tam$  ||  $array[2*i] > array[2*i+1]$ ) {  
        filho =  $2*i$ ;  
    } else {  
        filho =  $2*i + 1$ ;  
    }  
    return filho;  
}
```



Exercício Resolvido

• Implemente os métodos:

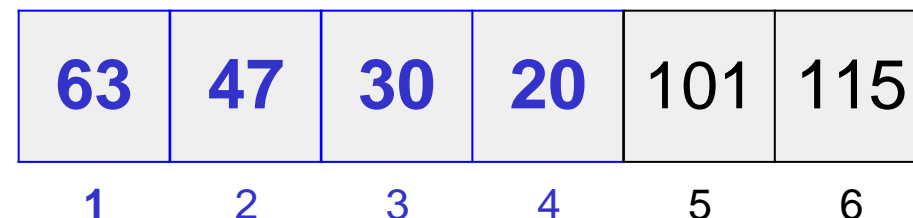
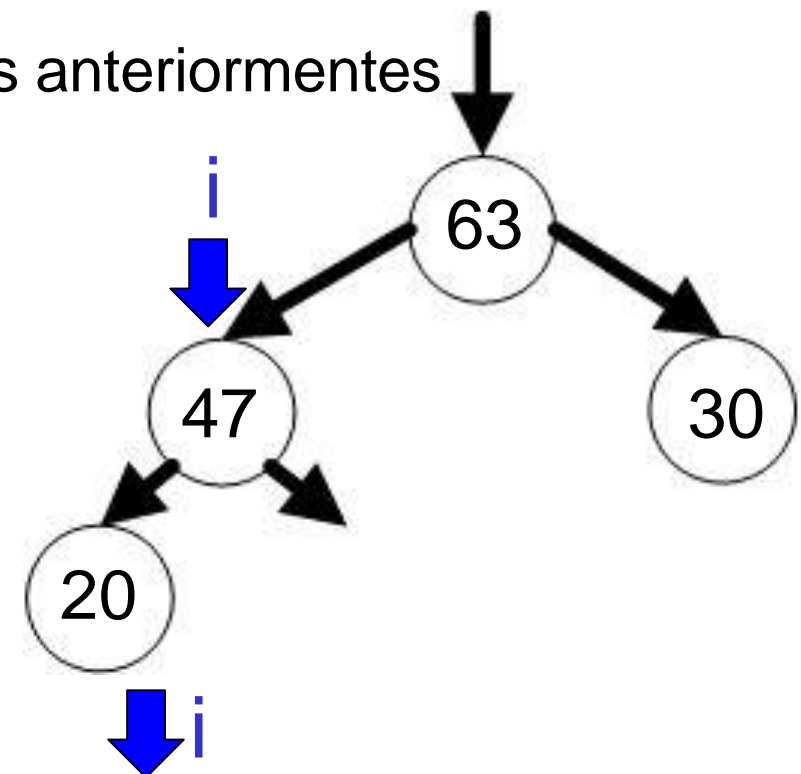
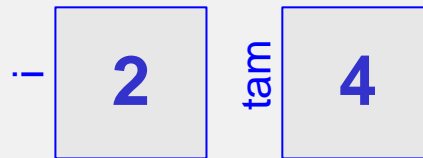
int GetMaiorFilho(**int** i, **int** tam) e

boolean HasFilho(**int** i, **int** tam) apresentados anteriormente

```

int GetMaiorFilho(int i, int tam){
    int filho;
    if ( $2*i == tam$  ||  $array[2*i] > array[2*i+1]$ ) {
        filho =  $2*i$ ;
    } else {
        filho =  $2*i + 1$ ;
    }
    return filho;
}

```



Exercício Resolvido

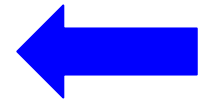
- Implemente os métodos:

int GetMaiorFilho(**int** i, **int** tam) e

boolean HasFilho(**int** i, **int** tam) apresentados anteriormente

```
boolean HasFilho(int i, int tam){  
    return (i <= (tam/2));  
}
```


- Definição de Heap
- Funcionamento básico
- Algoritmo
- **Análise do número de comparações e movimentações**



Análise do Número de Comparações

- As operações de inserção e remoção podem percorrer um ramo completo da árvore, com comparações e trocas em cada nó
- O pior caso para os números de comparações ou trocas depende da altura da árvore que será $\lceil \lg(n) \rceil$ (árvore balanceada)
- Assim, no pior caso, os números de comparações ou trocas serão $\Theta(\lg(n))$

Algoritmo

```
void Heapsort(int[] array, int n) {  
  
    int tam;  
    for (tam = 2; tam <= n; tam++){  
        Construir(array,tam);  
    }  
  
    //Ordenacao propriamente dita  
    tam = n;  
    while (tam > 1){  
        swap(1, tam--);  
        Reconstruir(array,tam);  
    }  
}
```

Análise do Número de Comparações

- **Primeiro passo:** criamos o heap através de $(n-1)$ inserções

```
int tam;
for (tam = 2; tam <= n; tam++){
    Construir(array,tam);
}
```

- **Segundo passo:** ordenamos o *array* através de $(n-1)$ remoções

```
//Ordenacao propriamente dita
tam = n;
while (tam > 1){
    swap(1, tam--);
    Reconstruir(array,tam);
}
```

- **Terceiro passo:** no pior caso, cada inserção/remoção percorre todos os elementos de um galho da árvore cuja altura é $\Theta(\lg(n))$, logo:

$$[(n-1) * \Theta(\lg(n))] + [(n-1) * \Theta(\lg(n))] = \Theta(n * \lg(n))$$

Análise do Número de Comparações

• **Quarto passo:** no melhor caso, temos que:

- cada inserção (chamada do Construir) faz somente uma comparação

```
void Construir(int[] array, int tam){  
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){  
        swap(i, i/2);  
    }  
}
```

- cada remoção continua percorrendo todos (ou quase todos) os elementos de um galho da árvore
- Logo:

$$[(n-1) * 1] + [(n-1) * \Theta(\lg(n))] = \Theta(n * \lg(n))$$

Análise do Número de Movimentações

- **Primeiro passo:** *swap* faz três movimentações
- **Segundo passo:** no pior caso, Construir / *Reconstruir* fazem $\Theta(\lg n)$ *swaps*

```
void Construir(int[] array, int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

- Logo,
- $$[(n-1) * \Theta(\lg(n)) * 3] + [(n-1) * \Theta(\lg(n)) * 3] = \Theta(n * \lg(n))$$

```
void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
```

Análise do Número de Movimentações

- **Terceiro passo:** no melhor caso, o Construir faz zero swaps e o Reconstruir, continua fazendo $\Theta(\lg n)$ swaps

```
void Construir(int[] array, int tam){
    for (int i = tam; i > 1 && array[i] > array[i/2]; i /= 2){
        swap(i, i/2);
    }
}
```

- Logo,

$$[0 * \Theta(\lg(n)) * 3] + [(n-1) * \Theta(\lg(n)) * 3] = \Theta(n * \lg(n))$$

```
void Reconstruir(int[] array, int tam){
    int i = 1;
    while (HasFilho(i, tam) == true){
        int filho = GetMaiorFilho(i, tam);
        if (array[i] < array[filho]) {
            swap(i, filho);
            i = filho;
        } else {
            i = tam;
        }
    }
}
```